Taller 7, Floyd's Tortoise

Miguel Daniel Ruiz Silva, Camila Andrea Galindo Ruiz,Daniel Felipe Gonzalez Beltran

I. Introduccion

La detección de ciclos en una lista enlazada y la búsqueda de números repetidos en una lista son tareas fundamentales en el ámbito de la programación y la estructura de datos. Estas operaciones son esenciales para garantizar la integridad y eficiencia de muchos sistemas informáticos. Para abordar estos desafíos de manera eficiente, el algoritmo de Floyd's Tortoise and Hare se ha convertido en una herramienta invaluable.

El objetivo de este proyecto es no solo comprender profundamente el algoritmo de Floyd's Tortoise and Hare, sino también aplicarlo de manera efectiva para detectar ciclos en una lista enlazada. Además, se busca realizar esta detección y buscar números repetidos en una lista con una complejidad menor a $O(n^2)$, lo que implica una optimización crucial en términos de tiempo de ejecución.

De la misma forma busca integrar estas funcionalidades en un microservicio existente. Esto se logrará mediante la creación de un nuevo endpoint que permitirá a los usuarios enviar un array en el cuerpo de una solicitud JSON para que el algoritmo realice las operaciones de detección de ciclos y búsqueda de números repetidos de manera eficiente.

II. implementación

El código contiene una clase 'Nodo' que representa los nodos de la lista enlazada, cada uno con un valor y una referencia al siguiente nodo. La función '$detectar - ciclo$' toma como entrada un nodo que representa el inicio de la lista y utiliza el algoritmo de Floyd's Tortoise and Hare para detectar ciclos en la lista. Se inicializan dos punteros, 'tortuga' y 'liebre', al inicio de la lista enlazada, y se ejecuta el algoritmo en dos fases. En la primera fase, 'tortuga' avanza un nodo a la vez y 'liebre' avanza dos nodos a la vez. Si existe un ciclo, eventualmente 'tortuga' y 'liebre' se encontrarán en el mismo nodo. En la segunda fase, el puntero 'liebre' se reinicia al inicio de la lista y ambos punteros avanzan un nodo a la vez hasta que se vuelvan a encontrar, identificando así el inicio del ciclo. La función imprime la representación de los nodos 'tortuga' y 'liebre' cuando se encuentran, proporcionando información sobre el ciclo, y retorna 'False' si no se detecta ningún ciclo.

III. pruebas

Algoritmo 1 - Floyd's Tortoise

En el siguiente fragmento de código, primero se definen cinco objetos de tipo Nodo (nodo1, nodo2, nodo3, nodo4, nodo5) y luego se establece una relación de ciclo conectando el último nodo (nodo5) nuevamente al segundo nodo (nodo2). Finalmente, se llama a la función detectar ciclo(nodo1) pasando el primer nodo (nodo1) como argumento, lo que inicia la ejecución de la función para detectar y verificar la existencia de un ciclo en la lista enlazada.

```
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
```

Fig. 1. Creacion de nodos y enlaces de nodos

```
<nodo 2> <nodo 2>
False
```

Fig. 2. Prueba del algoritmo Floyd's Tortoise

Algoritmo 2 - Búsqueda de números repetidos.

La función `encontrar-numero-repetido` implementa el algoritmo de la Tortuga de Floyd para encontrar el número repetido en la secuencia `nums`. La secuencia de prueba es $[3, 1, 3, 4, 2]$, y al ejecutar la función, se espera que identifique y devuelva el número repetido en la secuencia, que en este caso es el número 3. La línea de código que imprime el resultado muestra el número repetido como "El número repetido es: 3".

```
numbers = [1, 3, 4, 2, 2]
```

Fig. 3. Prueba del algoritmo de búsqueda de números repetidos.

```
El numero repetido es 2
```

Fig. 4. Prueba del algoritmo de búsqueda de números repetidos.

IV. La integración en el microservicio

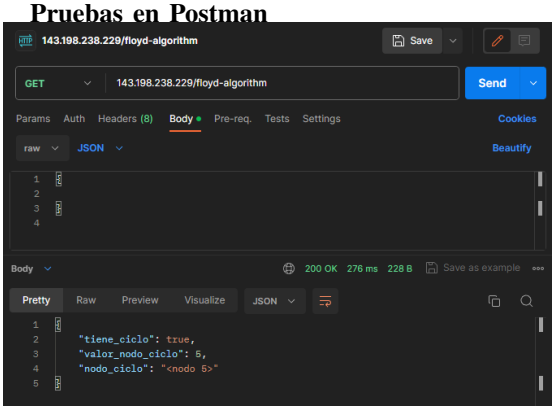
Se realiza la integración al microservicio,una vez obtenido y probado en ejecución el código se hacen pequeños cambios a cada uno para lograr integrarlo dentro del servidor y lograr realizar pruebas dentro de Postman.

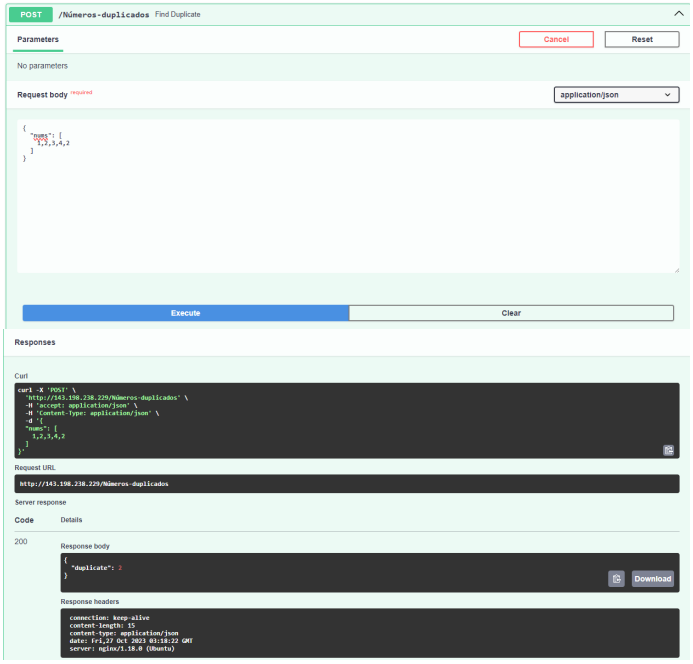
```python
class Numbers(BaseModel):
    nums: List[int]
```

```python
@app.get("/floyd-algorithm")
def floyd_algorithm():

    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

    nodo_ciclo = detectar_ciclo(nodo1)
    if nodo_ciclo:
        return {"tiene_ciclo": True, "valor_nodo_ciclo": nodo_ciclo.valor, "nodo_ciclo": str(nodo_ciclo)}
    else:
        return {"tiene_ciclo": False}
```

```python
@app.post("/Números-duplicados")
def find_duplicate(nums: Numbers):
    nums = nums.nums
    tortoise = hare = nums[0]
    while True:
        tortoise = nums[tortoise]
        hare = nums[nums[hare]]
        if tortoise == hare:
            break

    hare = nums[0]
    while tortoise != hare:
        tortoise = nums[tortoise]
        hare = nums[hare]

    return {"duplicate": hare}
```

Pruebas en Postman





V. Algoritmo de Floyd's Tortoise

El algoritmo de búsqueda de ciclos de Floyd, también conocido como el algoritmo Hare-Tortoise, emplea dos punteros, uno rápido (liebre) y otro lento (tortuga), para detectar ciclos en una lista enlazada. El puntero rápido avanza dos veces más rápido que el lento, lo que permite detectar ciclos mediante la igualdad de estos punteros. Si no hay ciclos, el puntero rápido alcanzará el final de la lista, mientras que, en presencia de un ciclo, los punteros se encontrarán en algún punto del ciclo. Una vez detectado el ciclo, se reinicia el puntero rápido al principio y se mueve tanto el puntero rápido como el lento un nodo a la vez para determinar el inicio del ciclo en el punto de encuentro. Este enfoque eficiente ofrece una valiosa herramienta para la detección de ciclos en listas enlazadas con una complejidad de tiempo proporcional al tamaño del ciclo.

Análisis de complejidad

El algoritmo de Floyd's Tortoise and Hare es altamente eficiente en términos de complejidad temporal, con una complejidad lineal O(n), donde "n" es el número de nodos en la lista enlazada, y requiere un espacio constante, O(1), independientemente del tamaño de la lista. Esto lo hace sobresalir en comparación con otros métodos para detectar ciclos en listas enlazadas que pueden tener complejidades temporales más altas y dependencia de estructuras de datos adicionales, lo que aumenta la complejidad espacial. En resumen, el algoritmo de Floyd's Tortoise and Hare es una elección eficiente y preferida para detectar ciclos en listas enlazadas en aplicaciones donde la eficiencia en tiempo y espacio es esencial.

VI. Busqueda de numeros repetidos usando el algoritmo de Floyd's Tortoise

Para encontrar números repetidos en una secuencia de números o en un arreglo. Se divide en dos fases: en la primera fase, dos punteros, uno que avanza una

posición y otro que avanza dos posiciones, exploran la secuencia hasta encontrar un punto de encuentro, que está dentro del ciclo de números repetidos. En la segunda fase, se restablece uno de los punteros al inicio de la secuencia, y ambos avanzan una posición a la vez hasta encontrarse nuevamente, esta vez en el número repetido.

Analisis de complejidad Su complejidad lineal, es decir, O(n), donde ”n” es el tamaño de la secuencia o arreglo en el que estás buscando números repetidos. Esto significa que el algoritmo es muy eficiente en términos de tiempo comparado con otros algoritmos, ya que no depende del tamaño de la secuencia de manera exponencial o cuadrática

VII. Repositorio

1.

$https://github.com/DanGonCol/Taller-7-Floyds-Tortoise$

2. codigo fuente del punto 2
3. codigo fuente del punto 3
4. ieee

VIII. Referencias

Oscar Mendez. (2023). Data Structures: Floyd’s Tortoise and Hare Algorithm Implementation. GitLab. URL:

$https://gitlab.com/konrad_lorenz/data_structures/floydstortoise$

GeeksforGeeks. (2023). Floyd’s Cycle Finding Algorithm. GeeksforGeeks. URL:

$https://www.geeksforgeeks.org/floyds-cycle-finding-algorithm/$

Oscar Mendez. (2023). Configuracion del servidor y despliegue del servicio. YouTube. URL:

$https://www.youtube.com/watch?v=YGSQgxBnb1k$