entrega-ejercicio-03

November 4, 2024

1 Entrega 3

[ ]:
```
#3. Añadir comentarios al código que expliquen los pasos del algoritmo

# Clase Nodo
class Node:
  def __init__(self, data):
    #Almacena el valor del nodo
    self.data = data
    self.next = None

# Clase ListaEnlazada. Contiene un objeto nodo
class LinkedList:
  def __init__(self):
    self.head = None


# Función de la clase LinkedList para eliminar un nodo en una posición␣
 ↪específica
def deleteNode(self, position):
  #Si la lista esta vacía no hay nada que eliminar devuelve return y terminamos
  if self.head is None:
    return

  #Si se quiere eliminar el primer nodo "position == 0", actualizamos el primer
  #nodo con el valor del siguiente nodo
  #Y devolvemos el nuevo primer nodo
  if position == 0:
    self.head = self.head.next
    return self.head
  #Creamos la variable index y la inicializamos con valor 0
  index = 0
  #Creamos la variable current y la inicializamos apuntando al primer nodo de␣
 ↪la lista
  current = self.head
  prev = self.head
  temp = self.head
```

```
  #Cuando current == None para
  while current is not None:
    #Si index es igual a la posición que se quiere eliminar
    if index == position:
    #Damos a temp el siguiente nodo al que queremos borrar y salimos
      temp = current.next
      break
    #Si index/=position actualizamos el valor de prev
    prev = current
    #Actualizamos el valor de current por el siguiente nodo
    current = current.next
    #sumamos 1 al valor index
    index +=1
    #Apuntamos al nodo temp
  prev.next = temp
  #devolvemos prev el nodo
  return prev
```

Ejercicio: estudiar el siguiente algoritimo sobre la estructura de datos Lista Enlazada.

1. ¿Cuál es la operación que realiza la función sobre la clase LinkedList? Como indica el nombre de la función, borra un nodo de una posición que se le pasa a la función por parámetros.
2. ¿Qué coste en tiempo tiene, en función del número N de elementos de la lista? El coste en tiempo de la función es de **O(N)** donde N es el número de nodos que se ha de recorrer para borrar el nodo que le indicamos a la función.

4. Programar una función principal que utilice la función para una aplicación en una lista ejemplo a elegir por el alumno. Indicación: utilizar las clases Node y LinkedList de los ejemplos de clase

[ ]:
```
class Node:
  def __init__(self, data):
    self.data = data
    self.next = None

class LinkedList:
  def __init__(self):
    self.head = None

  def printList(self):
    temp = self.head
    while temp:
      print(temp.data, end=" -> ")
      temp = temp.next
    print("None")

  def deleteNode(self, position):
```

```
    if self.head is None:
      return

    if position == 0:
      self.head = self.head.next
      return self.head

    index = 0
    current = self.head
    prev = self.head
    temp = self.head

    while current is not None:
      if index == position:
        temp = current.next
        break
      prev = current
      current = current.next
      index += 1
    prev.next = temp

# Función principal
if __name__ == '__main__':
  # Creamos la lista enlazada y añadimos algunos nodos
  lista_enlazada = LinkedList()
  lista_enlazada.head = Node(1)
  second = Node(2)
  third = Node(3)

  # Enlazamos los nodos: cabeza -> segundo -> tercero
  lista_enlazada.head.next = second
  second.next = third

  print("Lista original:")
  lista_enlazada.printList()

  # Borramos el nodo en la posición 1 (segundo nodo)
  lista_enlazada.deleteNode(1)

  print("Lista después de eliminar el nodo en posición 1:")
  lista_enlazada.printList()
```

```
Lista original:
1 -> 2 -> 3 -> None
Lista después de eliminar el nodo en posición 1:
1 -> 3 -> None
```

[ ]:

[ ]:
```python
# Algoritmo recursivo para el cálculo de la serie de Fibonacci, optimizado
def fibonacci(n, second_last, last):
    if n - 1 == 0:
        return second_last
    else:
        new_last = second_last + last
        second_last = last
        return fibonacci(n - 1, second_last, new_last)

if __name__ == "__main__":
    print(fibonacci(21, 0, 1))
```

```
6765
```

Ejercicio: Consideramos el siguiente algoritmo:

- **¿Cuál es la serie de números que calcula el algoritmo?**

El algoritmo calcula el número indicado de la serie fibonacci que pasamos por parametros con los valores iniciales 0 y 1.

- **¿Se trata de un algoritmo iterativo o recursivo?**

El algoritmo se llama a si mismo hasta que llega a la serie indicada. es recursivo

- **¿Cuál es el orden del coste en tiempo de este método?**

Este algoritmo recursivo optimizado tiene un coste en tiempo de O(N), donde N es el número de la serie que queremos calcular.

- **¿Es más eficiente o más costoso, respecto el algoritmo recursivo visto en clase para el cálculo de la serie?**

En comparación con los 3 algoritmos utilizados para calcular la serie de Fibonacci de la primera entrega, este es más eficiente en cuanto a tiempo al algoritmo que utilizaba un array y al que utilizaba dos variables, pero es menos eficiente que el algoritmo que usa matrices para calcular la serie Fibonacci O(logn)