Estructura de Datos y Algoritmos. **Grado en Ingeniería Informática** **Doble Grado en Ingeniería Informática y Administración de Empresas** **Universidad Carlos III de Madrid** **Caso Práctico. CURSO 2021-2022**	uc3m | Universidad Carlos III de Madrid

Enunciado caso práctico

El caso práctico se compone de tres fases. Este documento describe el enunciado de la primera fase. Los enunciados de la fase 2 y 3 se publicarán en las próximas semanas.

Fase 1 – TAD Lineales

Dada la clase SList, implementación del TAD Lista usando una única referencia al primer nodo de la lista, se pide implementar una subclase de la clase SList, que llamaremos SList2, con las siguientes funciones:

1. **sumLastN(n):** función que toma un número entero n y devuelve la suma de los n últimos nodos de la lista invocante. Si n<0, la función debe devolver None. Si el valor de n es mayor que el tamaño de la lista, la función debe devolver la suma de todos los elementos de la lista. Veamos algunos ejemplos:

***Entrada**: self: 10->6->8->4->12, n = -1*
***Salida**: None*
***Explicación**: Como n<0, devolvemos None, tal y como dice el enunciado.*

***Entrada**: self: 10->6->8->4->12, n = 0*
***Salida**: 0*
***Explicación**: La suma de los n=0 últimos elementos de la lista es 0 (no estamos sumando ningún número)*

***Entrada**: 10->6->8->**4->12**, n = 2*
***Salida**: 16*

***Explicación**: La suma de los n=2 últimos elementos de la lista es 4+12=16.*

***Entrada**: 10->6->8->4->12, n = 4*
***Salida**: 30*
***Explicación**: La suma de los n=4 últimos elementos de la lista es 6+8+4+12=30.*

***Entrada**: 10->6->8->4->12, n = 5*
***Salida**: 40*
***Explicación**: La suma de los n=5 últimos elementos (son todos los elementos) de la lista es 10+6+8+4+12=40.*

***Entrada**: 10->6->8->4->12, n = 8*
***Salida**: 40*
***Explicación**: La suma de los n=8 últimos elementos (son todos los elementos) de la lista es 10+6+8+4+12=40.*

---

2. **insertMiddle(e):** inserta el elemento e en el medio de la lista invocante. Si el tamaño de la lista es par, el nuevo elemento deberá insertarse en la posición len(self)//2. Si el tamaño es impar, entonces el elemento se deberá insertar en la posición (len(self)+1)/2. Si la lista está vacía, simplemente añadimos el elemento (puedes utilizar el método addFirst o addLast de SList). Veamos algunos ejemplos:

   ***Entrada:** self: 1->2->4->5, e=3*
   ***Qué debe hacer la función:** self: 1->2->3->4->5*
   ***Explicación:** el tamaño de la lista es 4, par, el nuevo elemento se debe insertar en la posición 2.*

   ***Entrada:** self: 5->10->4->32->16, e=41*
   ***Qué debe hacer la función:** 5->10->4->**41**->32->16*
   ***Explicación:** el tamaño de la lista es 5, impar, el nuevo elemento se debe insertar en la posición (5+1)/2=3.*

3. **insertList()**. La función toma como parámetros un objeto de la clase SList2, *inputList*, y dos números enteros, *start* y *end*. La función debe eliminar todos los elementos de la lista invocante entre las posiciones start y end, e insertar los elementos de la lista *inputList* en su lugar. Recuerda que asumimos que el primer índice o posición de una colección (lista, etc) es siempre 0. La función no devuelve nada, simplemente modifica la lista invocante. La función debe comprobar que los parámetros *start* y *end* son correctos (start>=0, start<=end, end<len(self)). Veamos algunos ejemplos:

***Entrada**: self: 1->8 ->7->2->5->4->6->8, inputList: 101->102->103->104, start=2, end=4*

***Qué debe hacer la función**: Modificar la lista invocante para que contenga los siguientes elementos: self: 1->8->101->102->103->104->4->6->8.*

***Explicación:** De la lista self, se han eliminado los elementos que hay entre las posiciones 2 y 4, ambas inclusives: self: 1->8->**7->2->5**->4->6->8, y en su lugar, se han añadido todos los elementos de la lista inputList:*

*self: 1->8 ->**101->102->103->104**->4->6->8.*

***Entrada**: self: 1->8 ->7->2->5->4->6->8, inputList: 101->102->103->104, start=6, end=7*

***Qué debe hacer el método**: Modificar la lista invocante para que contenga los siguientes elementos: self: 1->8 ->7->2->5->4->101->102->103->104*

***Explicación:** De la lista self, se han eliminado los elementos que hay entre las posiciones 6 y 7, ambas inclusives: self: 1->8->7->2->5->4->**6->8**. y en su lugar, se han añadido todos los elementos de la lista inputList:*

*self: 1->8->7->2->5->4->**101->102->103->104**.*

***Entrada**: self: 1->8->7->2->5->4->6->8, inputList: 101->102->103->104, start=0, end=3*

***Qué debe hacer el método**: Modificar la lista invocante para que contenga los siguientes elementos: self: 101->102->103->104->5->4->6->8*

***Explicación:** De la lista self, se han eliminado los elementos que hay entre las posiciones 6 y 7, ambas inclusives: self: **1->8->7->2**->5->4->6->8. Después se han añadido todos los elementos de la lista inputList en su lugar:*

*self: **101->102->103->104**->5->4->6->8.*

4. **reverse(k)**: la función debe invertir los elementos de la lista invocante en grupos de k elementos. Si k<=1, no se realiza ninguna transformación. Si k>=len(self), se debe invertir la lista completa. Veamos algunos ejemplos:

***Entrada**: self: 1 -> 8 -> 7 -> 2 -> 5 -> 4 -> 6 -> 8, k=2*
***Qué hace la función:** self: 8->1 ->2-> 7 -> 4 -> 5 -> 8 -> 6*
Explicación:
El primer grupo de k=2 elementos es 1->8, su inversa es 8->1
El segundo grupo de k=2 elementos es 7->2, su inversa es 2->7
El tercer grupo de k=2 elementos es 4->5, su inversa es 5->4
El cuarto grupo de k=2 elementos es 6->8, su inversa es 8->6

***Entrada**: self: 1 -> 8 -> 7 -> 2 -> 5 -> 4 -> 6 -> 8, k=3*
***Qué hace la función**: self: 7->8->1->4->5->2->8 -> 6*
Como la última sublista 6->8 tiene una longitud menor que k=3, nos limitamos a invertirla.
Explicación:
El primer grupo de k=3 elementos es 1->8->7, su inversa es 7->8->1
El segundo grupo de k=3 elementos es 2->5->4, su inversa es 4->5->2
Ahora en la lista ya solo quedan dos elementos 6->8, su inversa es 6->8

***Entrada**: self: 1 -> 8 -> 7 -> 2 -> 5 -> 4 -> 6 -> 8, k=8 (o k>8)*
***Qué hace la función**: self: 8->6->4->5->2->7->8->1*
***Explicación:** Como k=len(self) (o k>len(self)), invertimos la lista completa*

***Entrada**: self: 1 -> 8 -> 7 -> 2 -> 5 -> 4 -> 6 -> 8, k=1 (o k<=0)*
***Qué hace la función**: self: 1 -> 8 -> 7 -> 2 -> 5 -> 4 -> 6 -> 8*
***Explicación:** Si k=1, debemos invertir los grupos de k=1 elementos. La inversa de una sublista de tamaño 1 es la misma sublista. Por tanto, en este caso, no es necesario hacer nada.*
Si k<=0, no hacemos nada.

---

5. **maximumPair**(): la función debe devolver el valor máximo de la suma de elementos equidistantes en una lista. Dos nodos, n y m, son equidistantes de ambos extremos si la distancia del nodo n al principio de la lista es la misma que la distancia del nodo m al final de la lista. Si la lista tiene un número par de elementos, por ejemplo k elementos, se tendrán que comparar la suma de k/2 pares de elementos. Sin embargo, si la lista tiene un número impar de elementos, el elemento que está en la mitad de la lista, no tiene un elemento equidistante. Este elemento central no se suma a ningún otro, pero su valor sí se tendrá en cuenta como si fuera la suma de un par de elementos equidistantes. Veamos algunos ejemplos:

***Entrada**: 1 -> 8 -> 7 -> 2 -> 5 -> 4*
***Salida**: 13*
Explicación**: **1 -> 8 -> 7 -> 2 -> 5 -> 4
1+4=5, 8+5=13, 7+2=9, max(5, 13, 9) =13

***Entrada**: 1 -> 8 -> 7 ->10 -> 2 -> 5 -> 4, lista tamaño impar*
***Salida**: 13*
Explicación**: **1 -> 8 -> 7 -> 10 -> 2 -> 5 -> 4
1+4=5, 8+5=13, 10, 7+2=9, max(5, 13, 10, 9) =13

***Entrada**: 1 -> 8 -> 7 ->30 -> 2 -> 5 -> 4, lista tamaño impar*
***Salida**: 30*
Explicación**: **1 -> 8 -> 7 -> 30 -> 2 -> 5 -> 4
1+4=5, 8+5=13, 30, 7+2=9, max(5, 13, 30, 9) =30

Normas:

1. Una solución se considera correcta si es robusta (no tiene errores para ninguna entrada), es correcta (resuelve el problema) y es eficiente (pueden existir varias soluciones, debes tratar de buscar siempre la solución más eficiente)
2. En la implementación del caso práctico, no está permitido utilizar estructuras de Python como Listas, Queues o Diccionarios, etc. (Sin embargo, en el unittest si estamos utilizando listas de python para testear las soluciones).
3. El caso práctico debe ser realizado por un grupo formado por dos miembros (ambos deben pertenecer al mismo grupo reducido). En ningún caso se permitirá grupos con más de dos miembros. Tampoco se permiten grupos individuales ya que una de las competencias a evaluar será el trabajo en equipo. Si no tienes compañero, por favor, envía un correo a tu profesor de prácticas.
4. Junto con el enunciado de esta fase, se entrega al estudiante lo siguientes ficheros:

    a. El fichero **slist.py** que contiene la implementación de las clases SList lista simplemente enlazada y nodo simple enlazado. La clase SList solo almacena la referencia al primer nodo de la lista y su tamaño. **Este fichero no puede ser modificado en ningún caso. Si detectas algún error, por favor comunícaselo a tu profesor.**
    b. El fichero phase1.py que tendrás que completar para elaborar la solución a esta fase. El fichero ya contiene algo de código que te ayudará a empezar.
    c. El fichero unittest-phase1.py contiene los tests necesarios para evaluar cada una de las funciones propuestas en esta fase. **Este fichero no puede ser modificado en ningún caso. Si detectas algún error, por favor comunícaselo a tu profesor.** Te recomendamos que inicialmente no ejecutes todo el test, sino que te centres en testear una función. Por ejemplo, deberías empezar testeando la función *sumLastN* (es la función más sencilla). Te aconsejamos que comentes los métodos tests de las otras funciones. Según vayas avanzando en la implementación, podrás ejecutar todos los métodos test en la misma ejecución.
5. Modo de entrega: En el grupo reducido de aula global, se publicará una tarea titulada '**Entrega Fase 1**'. La fecha de Entrega para esta primera fase será el **28 de marzo, 9.00 am**.
6. Formato de entrega: un zip cuyo nombre está formado por los dos NIAS de los estudiantes que forman el grupo, separados por un guión. Por ejemplo, *10001234-10005678.zip.* Únicamente uno de los dos miembros, será el

encargado de subir la solución del grupo a la tarea. El zip deberá contener los ficheros: slist.py, unitest_phase1.py y phase1.py.

7. **Defensa**: se celebrará durante la clase presencial del **lunes 28 de marzo**. La defensa del caso práctico es un examen oral. La asistencia es obligatoria. Si algún alumno no asiste, su calificación en el caso práctico será NP. Durante esta defensa, el profesor planteará a cada miembro del equipo una serie de preguntas que deberá responder de forma individual. Cada miembro del equipo debe ser capaz de discutir cualquiera de las decisiones tomadas en el diseño e implementación de cualquiera de las funcionalidades descritas en el caso práctico. La nota final del caso práctico estará condicionada por la nota obtenida en la defensa. Si un alumno no es capaz de discutir y defender ciertas decisiones, no será calificado con respecto a estas, aunque hayan sido correctamente implementadas.
8. Se recomienda seguir las recomendaciones descritas en Zen of Python (https://www.python.org/dev/peps/pep-0020/) y la guía de estilo (https://www.python.org/dev/peps/pep-0008/) publicada en la página oficial de Python.