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

Fase 1 – Listas

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 los siguientes métodos:

1. **sumLastN(n):** método que toma un número entero n y devuelve la suma de los n últimos nodos de la lista invocante. Si n<0, el método debe devolver None. Si el valor de n es mayor que el tamaño de la lista, el método 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). **Sin embargo, no está permitido utilizar el método insertAt**. 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()**. El método toma como parámetros un objeto de la clase SList2, *inputList*, y dos números enteros, *start* y *end*. El método 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. El método no devuelve nada, simplemente modifica la lista invocante. El método 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 el método**: 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)**: el método 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**(): el método 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:

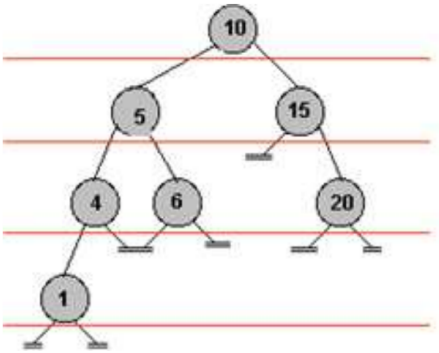
***Entrada**: 1 -> 8 -> 7 -> 2 -> 5 -> 4*
***Salida**: 13*
Explicación**: **1 -> 8 -> 7 -> 2 -> 5 -> 4
1+4=5, 8+5=13, 7+2=9, max(5, 13, 9) =13

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

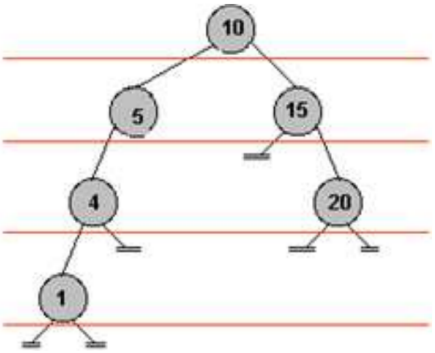
Entrada: *1 -> 8 -> 7 ->30 -> 2 -> 5 -> 4, lista tamaño impar*
Salida: *30*
Explicación: ***1 -> 8 -> 7 -> 30 -> 2 -> 5 -> 4***
1+4=5, 8+5=13, 30, 7+2=9, max(5, 13, 30, 9) =30

Fase 2 - Estructuras de datos no lineales ADT (árboles AVL)

Un árbol AVL es un árbol de búsqueda binario, en el que las alturas de los dos subárboles hijos de cualquier nodo difieren como máximo en uno. Cualquier árbol de búsqueda binaria T que satisfaga la propiedad de equilibrio de altura se dice que es un árbol AVL, llamado así por las iniciales de sus inventores: Adel'son-Vel'skii y Landis.



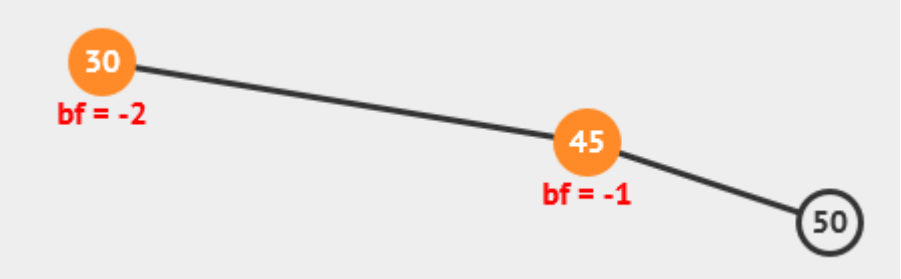
Un árbol de búsqueda equilibrado AVL



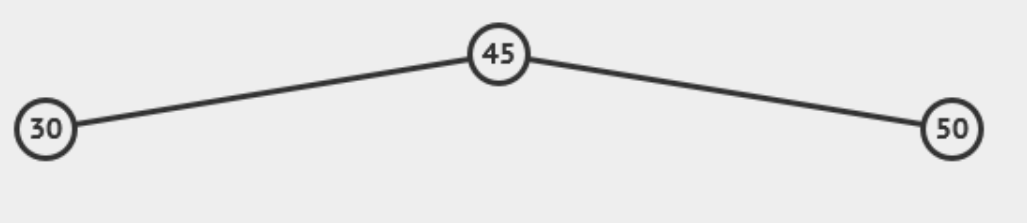
no es AVL porque la altura izquierda y la altura derecha del nodo 5 difieren en 2.

Los árboles AVL deben mantenerse equilibrados. Por tanto, después de una inserción de un nuevo nodo, el algoritmo de insertar deberá comprobar si en la rama donde se ha insertado el nuevo nodo, algún nodo ha quedado desequilibrado, y tendrá que equilibrarlo. De igual forma, después de borrar un nodo, el algoritmo de borrar deberá comprobar si en la rama donde se ha eliminado el nodo, algún nodo ha quedado desequilibrado, y tendrá que equilibrarlo.
Hay cuatro casos que hay que considerar para equilibrar un nodo desequilibrado. Estos casos se muestran a continuación:

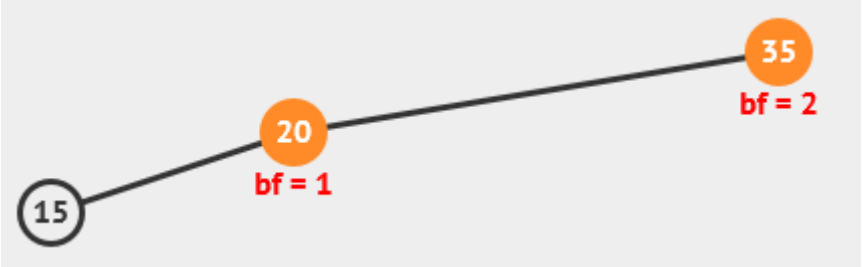
- **Rotación Simple Izquierda**: en la siguiente imagen, el nodo con elemento 30 está desequilibrado. Rotar un nodo nos va a permitir pasarlo de su rama más larga a su rama más corta. En este caso, la rama del nodo con elemento 30 es su rama derecha, por tanto, deberemos rotar el nodo a la rama izquierda. Su actual hijo derecho, nodo con elemento 45, se deberá convertir en la nueva raíz del árbol, y el nodo con elemento 30 pasará a ser su hijo izquierdo. El nodo con elemento 50, sigue siendo hijo derecho del nodo con elemento 45.



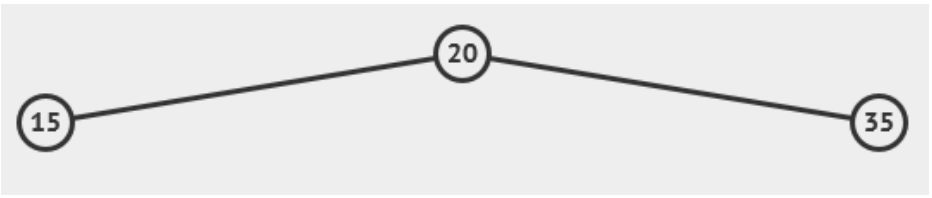
La rotación simple izquierda producirá el siguiente árbol, donde todos los nodos ya están equilibrados



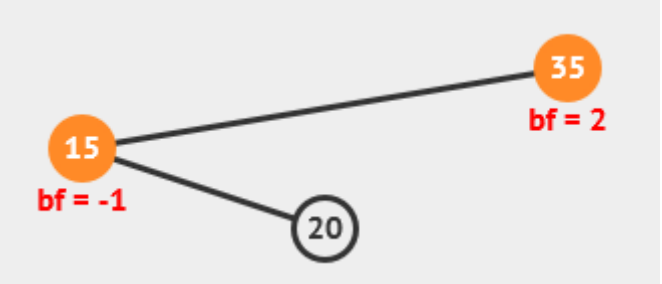
- **Rotación Simple Derecha**: como muestra la siguiente imagen, el nodo con elemento 35 está desequilibrado. La rama más larga que cuelga del nodo es su rama izquierda.



Si rotamos el nodo con elemento 35 a su rama derecha, estaremos consiguiendo reducir la longitud de dicha rama. Así, el nodo con elemento 35 girará a la derecha, convirtiéndose en el nuevo hijo derecho del nodo con elemento 20. Este nodo se convierte en la nueva raíz del árbol. El nodo con elemento 15 sigue siendo hijo izquierdo del nodo con elemento 20. La rotación simple derecha producirá el siguiente árbol, donde todos los nodos ya están equilibrados

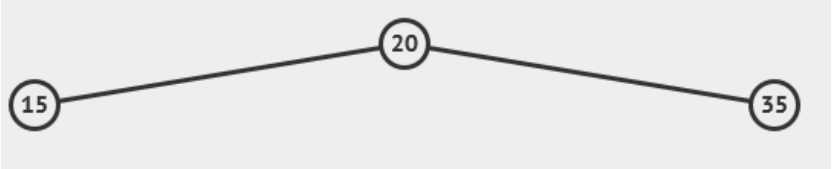


- **Rotación doble izquierda-derecha**. Estas rotaciones se aplican cuando la rama más larga que cuelga del nodo desequilibrado hace una especie de zigzag, como se muestra en la imagen siguiente. El nodo con elemento 35 está desbalanceado, y su rama más larga primero va a la izquierda, y luego a la derecha. En este tipo de casos, se deben realizar dos rotaciones:

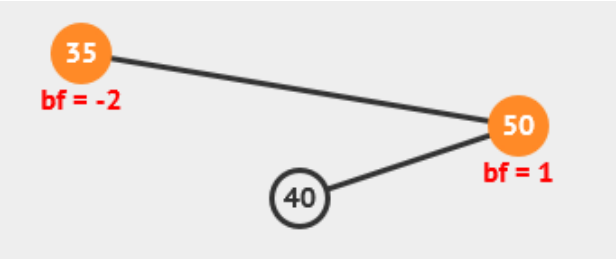


La primera rotación a aplicar será una rotación izquierda, para que el nodo con elemento 20, rote a la izquierda y se convierta en el nuevo hijo izquierdo del nodo desequilibrado (nodo con elemento 35), consiguiendo como resultado el siguiente árbol. El nodo con elemento 35 sigue estando desequilibrado.

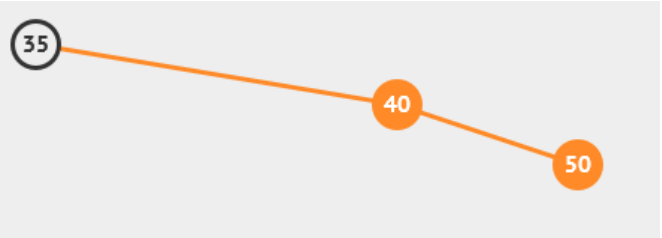
La segunda rotación a aplicar, ya la conocemos, es la rotación simple derecha, porque el nodo con el elemento 35 pasará a ser el subárbol derecho de la nueva raíz, el nodo con elemento 20. En el árbol resultante, ya están todos los nodos equilibrados.



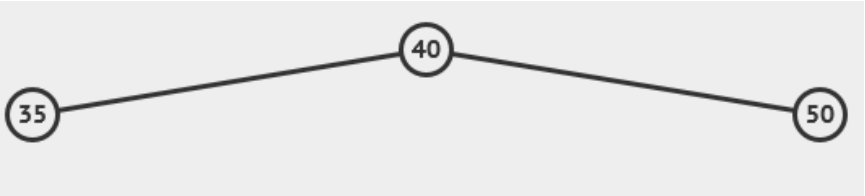
- **Rotación doble derecha-izquierda**. Como muestra la siguiente imagen, el nodo con elemento 35 está desequilibrado, y su rama más larga hace un zig-zag, primero se va a la derecha y luego a la izquierda. Aquí también deberemos aplicar una rotación doble.



  Primero aplicaremos una rotación a la derecha, para que el nodo con el elemento 40, rote a su derecha, convirtiéndose en el hijo derecho del nodo con elemento 35.



El nodo con elemento 35 sigue estando desequilibrado, pero ahora ya podemos aplicarle una rotación simple izquierda, rotando dicho nodo para que se convierta en el nuevo subárbol izquierdo del nodo con elemento 40, que a su vez se convertirá en la nueva raíz del árbol. En el árbol resultante, todos los nodos ya están equilibrados



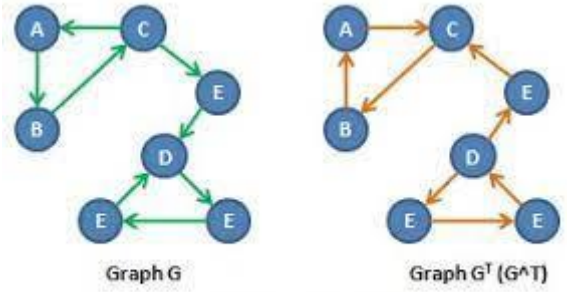
En la fase 2, tenéis que implementar un árbol AVL. Como se dijo anteriormente, un árbol AVL es un árbol binario de búsqueda con la propiedad que sus operaciones de inserción y borrado aseguran que el árbol resultante esté equilibrado.

Además de implementar los métodos insert y remove en la clase AVL, se recomienda la implementación de otras funciones que os ayuden a calcular el factor de equilibrio de un nodo, detectar si un nodo está desequilibrado y qué rotaciones deberían aplicarse, así como la implementación de las rotaciones específicas.

Fase 3 - Grafos

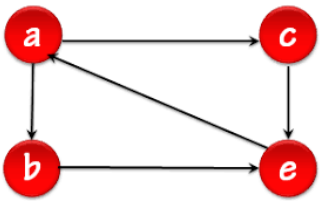
Dada la clase Graph2, hija de la clase Graph, implementa los siguientes métodos:

- **min_number_edges**: que reciba un par de vértices, start y end, y que devuelva el número mínimo de aristas entre ambos pares de vértices.
- **transpose**(): que devuelva el grafo transpuesto del grafo invocante (self). Un grafo transpuesto de un grafo G, tiene el mismo conjunto de vértices y las mismas aristas, pero con la orientación de las aristas invertidas. Es decir, si en el grafo G, existe una arista (u, v) (es decir, arista con origen u, y destino v), en el grafo transpuesto, la arista será (v, u), (es decir, origen v, destino u).

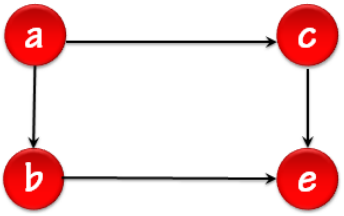


Graph G and its transpose G^T

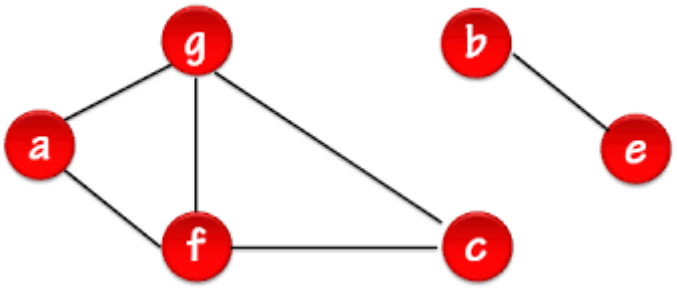
- **is_strongly_connected**(): devuelve True si el grafo (dirigido) es **fuertemente conexo**, y False en otro caso. Recuerda que un grafo dirigido es fuertemente conexo si existe un camino desde cualquier vértice a cualquier otro vértice. Por ejemplo, el grafo de la siguiente figura es un grafo dirigido fuertemente conexo, porque siempre es posible encontrar un camino para ir desde cualquier vértice a cualquier otro vértice.



Sin embargo, este otro grafo dirigido no está fuertemente conectado, porque por ejemplo, no es posible encontrar un camino desde c a b. De hecho, no es posible encontrar ningún camino que parta de e.



En el caso de grafos no dirigidos, la función devolverá True si el grafo es conexo y False en otro caso. Por ejemplo, para el siguiente grafo no dirigido, la función deberá devolver False porque es un grafo no conexo:



Sin embargo, si únicamente consideramos el subgrafo formado por los vértices a, f, c y g, la función devolverá True.

En la implementación de las tres fases, 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.