| Estructura de Datos y Algoritmos.<br>Ingeniería Informática<br>Universidad Carlos III de Madrid<br><br>Lab Case. CURSO 2018-2019 | uc3m \| Universidad Carlos III de Madrid |
|---|---|



# Creando una red social en la universidad.

La Universidad tiene un proyecto para crear una nueva red social para los estudiantes de la universidad mediante la integración de las redes sociales existentes administradas por departamentos y facultades.

El objetivo del ejercicio actual es definir e implementar las diferentes estructuras de datos y algoritmos que de soporte a la red social de acuerdo con los requisitos especificados en cada una de las siguientes fases.

## Fase 1

Teniendo en cuenta que, en una primera etapa, la red social incluirá usuarios de redes sociales ya existentes, se considera que estas redes se pueden representar con listas doblemente enlazadas. Se proporcionan las siguientes dos clases (**las cuales no se pueden modificar ni ampliar**).

- Una **Student** class que almacena información sobre cada alumno. Sus atributos son:
  - **email**: este es el correo electrónico para el alumno en la red social. Cada correo electrónico es único. Es decir, no puede haber diferentes alumnos con el mismo correo electrónico.
  - **city:** ciudad de residencia.
  - **campus:** (GETAFE, LEGANES, COLMENAREJO). Se ha definido un enum type[1], Campus, dentro de class Student.
  - **blocks**: Los estudiantes en la red pueden bloquear a otros estudiantes para prohibirles que se comuniquen con ellos. Este atributo almacena el número de veces que este estudiante ha sido bloqueado por otros usuarios.
  - **date_sign_in**: La fecha en la que el usuario se registró en la red social. Nosotros hemos usado la **java.time.LocalDate**[2] class para

[1] Tutorial for creation of a enum type:
https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html
[2] https://docs.oracle.com/javase/8/docs/api/java/time/LocalDate.html

representar datos. Esta class no almacena o representa una hora o zona horaria. Solo almacena el año, mes y día de una fecha.

- Una **StudentsList** class que implementa el IList interface. Contiene a todos los alumnos registrados en una red social. La red social no debe contener estudiantes repetidos. Por esta razón, se puede ver que los métodos *addLast*, *addFirst* and *insertAt* siempre verifica si el estudiante existe o no, antes de añadirlo a la lista.

En esta fase, se debe implementar una nueva clase, **ManageNetworkList,** con los siguientes métodos (estos métodos están también descritos en la interfaz IManageNetwork que se ha proporcionado):

1. Crear un método, denominado **union**, que une dos redes sociales en una sola red social. Es decir, el método toma como entrada dos objetos de la StudentsList y devuelve una nueva lista que contiene primero a los estudiantes de la primera lista, seguido por los estudiantes de la segunda lista.

2. Crear un método **getCampusCity** que toma como entrada una red social (es decir, un objeto de la StudentsList class) y un entero como parámetro opc tal que:
   - Si opc =1: el método debe devolver un objeto StudentsList que contenga todos los estudiantes que residen en la misma ciudad que el campus donde ellos están estudiando.
   - Si opc =2: el método debe devolver un objeto StudentsList que contenga todos los estudiantes que residen en diferentes ciudades a la del campus en el que estudian.

Nota: El orden en la lista resultante debe ser el mismo que en la lista de entrada.

3. Crear un método **locateByCity** que toma una red social (es decir, un objeto de la StudentsList class) y un nombre de ciudad como entrada, y devuelva una lista que contiene todos los estudiantes (es decir, un objeto de la StudentsList class) que viven en esa ciudad.

Nota: El orden en la lista resultante debe ser el mismo que en la lista de entrada.

4. Crear un método **orderBy** que toma como entrada una red social (es decir, un objeto de la StudentsList class) y un parámetro entero opc tal que:
   - Si opc=1, el método devuelve una nueva lista de estudiantes (es decir, un objeto de la StudentsList class) ordenados por orden ascendente según su email.
   - Si opc=2, el método devuelve una nueva lista de estudiantes (es decir, un objeto de la StudentsList class) ordenados por orden descendente según su email.

**Nota 1.** Se debe implementar un propio método de clasificación basado en algunos de los algoritmos ordenados (como burbuja, sort)[3].

[3] http://www.java2novice.com/java-sorting-algorithms/

**Note 2:** Recordad que no se puede modificar ni extender la StudentsList class. Por lo tanto, si se necesitase un método auxiliar que ayude a ordenar la lista, incluir el método en el **ManageNetwork class.**

**Nota 3.** La lista de entrada no se puede modificar. El método debe devolver una nueva lista donde se ordenan los estudiantes.

5. Crear un método **getStudentsByDateInterval**, el cual toma como entrada una red social (un objeto de la StudentsList class) y dos fechas (start y end, las cuales deben definirse como objetos java.time.LocalDate class), y devuelve una lista con todos los alumnos de la lista de entrada cuyas fechas de registro se encuentran en este intervalo de fechas. Ténganse en cuenta los siguientes comentarios:
    a. start <= end.
    b. Ambas fechas están incluidas en el intervalo.
    c. El orden en la lista resultante debe ser el mismo que en la lista de entrada.
6. Elaborar una tabla con las funciones Big-Oh para cada método de **ManageNetwork class**. Incluir una breve explicación por cada método.

## Fase 2

A medida que aumenta el número de estudiantes, el tiempo de acceso de búsquedas de estudiantes por nombre de usuario aumenta a límites inaceptables. Para mejorar el tiempo de acceso, se proporciona un árbol binario de búsqueda **StudentsTree** para almacenar estudiantes de una red social determinada (**que no se puede modificar!**) y la BSTNode class con los métodos para trabajar con los nodos de búsqueda binarios. La **StudentsTree** class implementa la interfaz **ITree**.

Crear una class, **ManageNetworkTree**, que incluya los siguientes métodos:

1. Crear un método, **copySocialNetwork**, que toma como entrada un objeto de la StudentsTree class y uno objeto de la StudentsList class (Fase 1) e inserta de la lista en el árbol. El método no debe devolver nada.
2. Crear un método **getOrderedList,** que toma como entrada un objeto de la StudentsTree class y devuelve un objeto de la StudentsList class que contiene todos los estudiantes ordenados por el email en el árbol. Para obtener esta lista, no se puede utilizar ningún algoritmo de clasificación (como burbuja, etc.), se debe recorrer el árbol de forma recursiva.

Sugerencia: puede usar un método auxiliar y recursivo, que tome como entrada un objeto de BSTNode y un objeto de StudentsList, y devuelva los estudiantes (ordenados por email) de este subárbol.

3. Crear un método **deleteByNumberOfBlocks** teniendo como entrada un objeto de StudentsTree y un entero n. El método debe eliminar a todos los estudiantes que tengan un número de bloqueos igual o mayor que n.

Sugerencia: se puede utilizar un método auxiliar y recursivo, que toma como entrada un objeto BSTNode y un entero n. Además, se puede utilizar el método remove del StudentsTree para eliminar los nodos.

4. Los miembros de la red social también pueden ser buscados por su fecha de inicio de sesión. ¿Es la StudentsTree class una estructura de datos eficiente para este tipo de buscadores?. Razona tu respuesta. Si se considera que no es eficiente, describir una estructura de datos más eficiente para dar soporte a este tipo de buscadores.

## Fase 3

Una de las capacidades de una red social es permitir que los usuarios sigan a sus amigos o personas que comparten contenidos interesantes. Existe la necesidad de proporcionar una estructura de datos que de soporte a esta capacidad. Como es de imaginar, la estructura de datos más adecuada es un grafo. En esta fase se quiere proporcionar una implementación de la red social basada en una estructura de datos de grafo. Esta debe ser denominada como **StudentGraph**. Para simplificar el ejercicio, el grafo solo tiene que almacenar los emails de los estudiantes (ver Nota). Hay que tener en cuenta que la universidad cuenta con más de 20 mil alumnos.

1. ¿Qué tipo de representación de grafo es la más adecuada para un número tan grande de usuarios posibles? Explica tu respuesta.
2. Crear un **constructor** para la StudentGraph teniendo como entrada un array con los emails de los estudiantes registrados en la red social. Cuando se crea un objeto de la StudentGraph class, no habrá ninguna relación entre los estudiantes. En otras palabras, el método del constructor solo almacena los emails de los estudiantes en el grafo, sin crear ninguna relación entre ellos.
3. Crear un método **addStuden**t que tome como entrada un email y lo añade a la red social.
4. Crear un método **areFriends** que tome como entrada dos estudiantes (emails) y cree una relación de amistad entre ellos. Tened en cuenta que la relación de amistad es una relación simétrica.
5. Crear un método **getDirectFriends** que, dado un estudiante (email) como entrada, devuelva un objeto de LinkedList<String> que contenga los emails de sus amigos directos.
6. Crear un método **suggestedFriends** tal que, dado un estudiante (email) como entrada, devuelva un objeto de LinkedList<String> que contenga los emails de sus potenciales amigos. Es decir, el método debe encontrar aquellos estudiantes conectados con el estudiante dado pero que no tengan un enlace directo.

Sugerencia: algunos de los algoritmos de recorrido de grafos pueden ser útiles para implementar la solución.

**Nota**: En la fase 3, se recomienda encarecidamente que se utilicen clases de Java Collections Framework (por ejemplo, LinkedList), en lugar de utilizar arrays o implementaciones de listas. Es una buena oportunidad para ampliar conocimientos. Así, por ejemplo, si finalmente decide implementar un grafo

basado en listas de adyacencia, podría usar un LinkedList de LinkedLists!, en lugar de usar un array de linkedlist (como hemos visto en nuestras clases de teoría). Pídele ayuda a tu maestro.

## Instrucciones generales:

1. La práctica de laboratorio debe ser realizada por un equipo de dos estudiantes (los cuales deben pertenecer al mismo grupo de laboratorio). En ningún caso, se permitirán equipos con más de dos miembros. Si no se tiene ningún compañero para llevar a cabo el caso de laboratorio, SE DEBE NOTIFICAR AL PROFESOR DE LABORATORIO LO ANTES POSIBLE.
2. Se proporciona un proyecto Java con varias interfaces y clases implementadas que **NO PUEDEN SER MODIFICADOS**:
    a. Fase 1:
        - Interfaces: *IList* (interface para una lista de estudiantes). *IManageNetworkList*, la cual define un conjunto de métodos que hay que implementar en la ManageNetworkList class.
        - Classes: *Studen*t class que contiene la información relativa al estudiante en una red social. La *DNode* y *StudentsList classes*, las cuales implementan un nodo doble y una lista doblemente enlazada que almacena objetos de estudiantes.
        - Junit Test: ManageNetworkListTest, que ayuda a validar si tu solución es correcta.
    b. Fase 2:
        - Interfaces: *IBSTNode*, *IBSTTree* interfaces para nodos y árbol binario de búsqueda respectivamente. El *IManageNetworkTre*e interface contiene un conjunto de métodos que hay que implementar en la *ManageNetworkTree class*.
        - Classes: *BSTNode*, *StudentsTre*e, las cuales son clases para un nodo y un árbol binario de búsqueda que almacena los estudiantes.
        - JUnit Test: *ManageNetworkTreeTest*, que ayuda a validar si tu solución es correcta.
    c. Fase 3:
        - Interface: *IManageNetworkGraph*, el cual define un conjunto de métodos que se deben implementar en la *ManageNetworkGraph* class.
        - Classes: *Adjacent*, es una clase auxiliary que almacena el índice del vértice.
        - JUnit Test: *ManageNetworkGraphTest*, que ayuda a validar si tu solución es correcta.

3. SOLO SE TIENEN QUE COMPLETAR LAS SIGUIENTES CLASES:
    a. Fase1: **ManageNetworkList**.
    b. Fase 2: **ManageNetworkTree**.
    c. Fase 3: **ManageNetworkGraph.**
4. Entregar antes del **6 de mayo a las a.m** a través de la plataforma AulaGlobal. Subir un zip con los siguientes archivos:
    a. Las clases: **ManageNetworkList.java, ManageNetworkTree.java, ManageNetworkGraph.java.**
    b. Un documento con las respuestas a las preguntas. Además, se puede incluir aquellos comentarios que se consideren necesarios sobre su solución.
    c. El nombre del zip debe ser la concatenación de los NIA de los estudiantes del grupo (por ejemplo, “10002354_10004532.zip”).
5. Las Junit test clases son muy importantes. La solución de la práctica debe pasar estos tests junit. Por lo tanto, se debe implementar las clases teniendo en cuenta estos test (**que nunca se podran modificar** !!!). **Aquellas soluciones que no tengan en cuenta las condiciones y restricciones de los tests Junit, no serán evaluadas**.
6. Sólo un miembro del equipo será el responsable de subir el archive zip a la plataforma AulaGlobal (en la tarea creada denominada "Lab Case Submission").
7. Es obligatorio que los dos miembros del grupo asistan a una sesión de defensa de la práctica, que está programada para la semana del 6 al 13 de mayo (se confirmará a través de AulaGlobal). Por esta razón, el profesor del laboratorio probará la solución de los alumnos utilizando los test JUnit (similares a los test Junit proporcionados). Un método se considerará correcto solo si su test Junit no falla. Además, el profesor de laboratorio podrá preguntar sobre algunos detalles de la implementación de la práctica. Ambos miembros del grupo deben participar en la implementación de todas las fases y tener un conocimiento profundo de ellas.