Estructura de Datos y Algoritmos. Ingeniería Informática Universidad Carlos III de Madrid Caso Práctico. CURSO 2019-2020	uc3m | Universidad Carlos III de Madrid

Aplicación para gestionar los repartos de Amazon.

Fase 1 - Estructuras lineales

En Amazon, los paquetes están identificados con la siguiente información:

- un identificador de paquete (por ejemplo, '*132-1352234-332344'*). El identificador debe serguir el siguiente formato XXX-XXXXXX-XXXXXX, donde X es cualquier dígito.
- dirección de entrega, que se descompone en nombre de la vía y número (por ejemplo, '*c/Federica Montseny nº7'*), obviando la información sobre el piso (para simplificar), y
- el código postal (por ejemplo, *28911*).

Amazon cuenta con una plantilla de repartidores. De cada repartidor, la información relevante es:

- el Id de repartidor (por ejemplo, '*R1000335'*). El identificador deberá seguir el siguiente formato: RXXXXXX, donde X es cualquier dígito.
- Apellidos, Nombre: (por ejemplo, '*Segura Bedmar, Isabel"*.
- su estado ('*activo*' o '*no activo*'). El estado 'no activo' sirve para identificar aquellos repartidores que están ausentes por diferentes causas.

Además, cada repartidor tiene asignado una o más zonas de reparto. Para simplificar el problema, vamos a suponer que cada zona se corresponde con un código postal. Así por ejemplo, el repartidor '*R1000335'*, podría tener asignadas las siguientes zonas de reparto: 28911, 28912, 28913, 28914. Una zona podría ser asignada a varios repartidores.

Cada día Amazon tiene que procesar los paquetes que deben ser repartidos ese día (aunque para simplificar el problema, no vamos a representar ninguna información relativa a las fechas). Como se ha dicho antes, cada paquete deberá ser asignado a alguno de los repartidores que cubren la zona del paquete. Si la zona del paquete no ha sido asignada a ningún repartidor, dicha zona se asignará a un repartidor en activo con menor número de zonas asignadas. De esta forma, cada repartidor al principio de su jornada tendrá un conjunto de paquetes asignados.

El repartidor deberá entregar sus paquetes en el mismo orden en que le fueron asignados. En la entrega del paquete, pueden ocurrir dos cosas: i) el paquete es entregado en el domicilio, o bien, ii) el paquete no se ha podido entregar. En el primer caso, los datos del paquete deberán ser registrados en una estructura de datos donde se almacenan los paquetes entregados correctamente. De cada paquete entregado correctamente, será suficiente con almacenar el Id del Pedido y el Id del repartidor. Además, dicho paquete deberá ser eliminado del conjunto de paquetes asignado al repartidor.

Si por el contrario el paquete no pudo ser entregado, el paquete deberá pasar a ser el último paquete asignado al repartidor (es decir, deberá ser entregado después del resto). Al tercer intento de entrega, el paquete será eliminado de la lista de paquetes asignados al repartidor y será registrado en estructura de datos que recoja los paquetes con alguna incidencia. En este caso, sólo será necesario almacenar el id del paquete, el id del repartidor y el motivo de la incidencia '*superado el número de intentos en la entrega*'.

Si un repartidor, por cualquier motivo, tiene que abandonar el reparto de los paquetes, cada uno de los paquetes pendientes debe ser reasignado a un repartidor activo que dé servicio a la zona del paquete. Si no hay ningún repartidor disponible, entonces el paquete deberá pasar a incidencias, identificando que el motivo de la incidencia es '*no repartidor disponible para su reparto"*.

A continuación se describen las diferentes tareas que debes realizar en esta práctica:

- Implementa las estructuras de datos necesarias para representar:
  - Una estructura de datos para representar la información relativa a un paquete. Su nombre debe ser **Paquete**.
  - Una estructura de datos que contenga la información de todos los paquetes que Amazon tiene que distribuir. Su nombre debe ser **Paquetes (Pedidos)**.
  - Una estructura de datos para representar la información relativa a un repartidor. Su nombre debe ser **Repartidor**. Recuerda que un repartidor tiene una o más zonas asignadas y un conjunto de paquetes a repartir.
  - Una estructura que almacene todas los repartidores de Amazon. Su nombre debe ser **Repartidores.**
  - Una estructura de datos para almacenar los paquetes que han sido correctamente entregados. Su nombre debe ser **Entregados**.
  - Una estructura de datos para almacenar los paquetes que no han podido ser entregados. Su nombre debe ser **Incidencias**.

Además, tendrás que implementar una clase principal, **GestionAmazon**, que permite simular la funcionalidad que describimos en los siguientes párrafos. Esta clase, entre otros, deberá incluir un atributo que represente todos los pedidos (paquetes), un atributo que represente todos los repartidores, así como atributos para representar los paquetes entregados correctamente y otro atributo para las

incidencias. La clase podría tener otros atributos. Esta clase debe incluir las siguientes funcionalidades:

- Implementa un método, **cargarPedidos**(), que reciba los datos necesarios para inicializar el objeto que almacena los pedidos que Amazon deberá enviar.
- Implementa un método, **cargarRepartidores**(), que reciba los datos necesarios para inicializar el objeto que almacena los repartidores de Amazon. En este método, los repartidores no tendrán ningún paquete asignado.
- Implementa un método, **mostrarRepartidores**, que muestre la lista de repartidores (identificador, estado, zonas y paquetes asignados) en orden alfabético ascendente (apellidos).
- Implementa un método, **asignarReparto()** que asigne cada pedido (paquete) a un repartidor. Recuerda que el repartidor debe estar activo y cubrir la zona del paquete. Si no hay ningún repartidor activo para esa zona, el paquete (y su zona) será asignado al repartidor activo con menor número de zonas.
- Escriba un método, **reparto()**, que permita simular la entrega de paquetes de todos los repartidores.
- Implementa un método, **entregaPaquetes(repartidor)**, que simule la entrega de los paquetes de un determinado repartidor. El método recibe un repartidor y debe procesar todos sus paquetes, comenzando siempre por el primer paquete y procesarlos en estricto orden. Para cada paquete a entregar, de forma aleatoria, se genera un valor (True o False) para indicar si el paquete ha sido entregado correctamente o no. Si el valor generado es True, el paquete deberá ser eliminado del conjunto de paquetes asignados a ese repartidor y registrado como entregado. Si por el contrario el valor generado es False, interpretaremos que el paquete no ha podido ser entregado, así que pasará a estar el último en el orden de entrega de los paquetes pendientes de entregar por ese repartidor. Si el repartidor ha intentado entregar 3 veces dicho paquete sin éxito, será eliminado del

reparto y registrado como incidencia. El método siempre debe mostrar para cada paquete procesado, el id del paquete y un mensaje informando si fue entregado correctamente, si queda pendiente (con el número de intentos realizados) o si es retirado. La entrega de paquetes para un repartidor debe continuar mientras el repartidor aún tenga paquetes por entregar.

- Escriba un método, **bajaRepartidor()**, que reciba un repartidor y lo ponga como no activo. Si el repartidor tenía algún paquete por entregar, deberá ser reasignado a algún repartidor activo que cubra la zona del paquete. Si no se encuentra ningún repartidor disponible, el paquete será registrado en incidencias.

Análisis de la Complejidad

Cada método debe incluir un comentario que indique su complejidad temporal, y explique brevemente el peor y mejor caso.

Fase 2 - Árboles Binarios de Búsqueda

Como se explicó en la fase anterior, para asignar un paquete a un determinado repartidor en activo, era necesario recorrer la lista de repartidores y buscar un repartidor en activo que cubra la zona del paquete. Esta operación tiene una complejidad O(n), donde n es el número de repartidores. En el peor de los casos será necesario recorrer toda la lista para encontrar un repartidor adecuado o bien para determinar que esa zona no es servida por ningún repartidor en activo.

El objetivo de esta fase es desarrollar una estructura de datos que permita que este tipo de búsqueda (por zona) tenga complejidad logarítmica. Además, dicha estructura de datos debe permitir agregar o eliminar nuevas zonas, también con una complejidad logarítmica. Recuerda que cada zona puede ser cubierta por varios repartidores.
En esta fase, sólo nos vamos a centrar en la representación de las zonas y sus repartidores asignados. Es decir, no vamos a implementar la funcionalidad relacionada con los paquetes, que ya implementamos en la fase 1. Además, para simplificar esta fase, vamos a considerar lo siguiente:

- Una zona está representada de forma única por un código postal. Una zona deberá estar asociada a sus repartidores.
- De cada repartidor sólo vamos a almacenar su nombre, es decir, no vamos a guardar ni su id ni tampoco los paquetes que tiene que repartir.

En concreto, la estructura de datos a desarrollar debe incluir los siguientes métodos:

- Implementa un método, **crearZona**, que permita añadir una nueva zona. El método recibe como parámetro un cp que identifica la zona. Inicialmente no tendrá ningún repartidor asignado. La complejidad de este método debe ser logarítmica.
- Implementa un método, **asignaRepartidor()**, que reciba como parámetro el cp que identifica la zona y el nombre del repartidor, y relaciona al repartidor con esa zona.

- Implementa un método, **repartidores**(), que reciba una cp y muestre sus repartidores ordenados alfabéticamente.
- Implementa un método, **borrarRepartidor()**, que elimine un repartidor de una determinada zona. El método recibe como parámetros el cp que identifica la zona y el nombre del repartidor.
- Implementa un método, **zonas**(), que devuelva la lista de todas las zonas ordenadas de forma ascendente. Cada zona debe incluir su lista de repartidores asignados, también ordenada de forma alfabética.
- Implementa un método, **borrarZona**, que dada una zona, la elimina y reasigna sus k repartidores a otras zonas. En concreto, los repartidores deberán ser repartidos entre las zonas anteriores (precedentes en la estructura) y siguientes (sucesoras en la estructura de datos) de la zona que va a ser eliminada. El reparto debería ser equitativo entre las zonas anteriores y siguientes, es decir, si la zona tiene k repartidores, k/2 deberían ir a las zonas predecesoras y k/2 a las zonas sucesoras. Deberás decidir cómo tratar el caso cuando k es impar. En un principio, a cada zona sólo se le asignará un único repartidor, salvo en el caso en el que no haya suficientes zonas anteriores y siguientes para alojar a los k repartidores. En este caso, los repartidores se deberán repartir de la forma más equitativa entre las zonas predecesoras y sucesoras, pero recuerda que la mitad siempre debe estar albergada en zonas predecesoras, y la otra mitad en las zonas sucesoras.
- Implementa un método, **esBalanceada()**, que compruebe si las zonas están almacenadas de forma balanceada en la estructura de datos. Debes basarte en el equilibrado perfecto, donde dada una zona, la diferencia entre el número de zonas anteriores y siguientes, debe ser como máximo de 1. Si la estructura no está equilibrada, entonces deberás balancearla siguiendo la estrategia de equilibrado perfecto (o en tamaño) explicado en clase. ¿Qué ventajas ofrece balancear la estructura?

Análisis de Complejidad: Cada método debe incluir al principio un comentario donde se indique su complejidad, también de su peor y mejor caso.

Fase 3 - Grafos

Un repartidor cada día recibe un **mapa** con los distintos puntos de entrega de su reparto. El objetivo de esta fase será desarrollar una estructura de datos que permita representar el mapa de reparto de un repartidor. Dicho mapa estará compuesto de **puntos de entrega** (que están definidos por el nombre de la vía, número y código postal) y las posibles conexiones entre los puntos de entrega. Si dos puntos de entrega están directamente conectados, el mapa también contendrá información sobre la distancia en km entre ambos puntos de entrega. Algunos puntos de entrega no están directamente conectados entre ellos, siendo necesario pasar por uno o varios puntos de entrega intermedios. La estructura de datos, **Mapa**, debe proporcionar la siguiente funcionalidad:

- Implementa un método, **anadirPuntoEntrega**(), que permitan añadir un nuevo punto de entrega.
- Implementa un método, **anadirConexion**(), que reciba dos puntos de entrega y un distancia, y establece una conexión entre ambos puntos de entrega.
- Implementa un método, **eliminarConexion**(), que reciba dos puntos de entrega, y elimine su conexión.

- Implementa un método, **__str__**(), que muestre todos los puntos de entrega y sus conexiones directas con otros puntos y sus distancias.
- Implementa un método, **estanConectados**(), que compruebe si dos puntos de entrega están **directamente conectados**. Si están conectados deberá devolver la distancia, en otro caso devolverá 0.
- Implementa un método, **generarRuta**(), que permite visitar todos los puntos de entrega. El método debe **devolver una lista de Python** que contiene los puntos de entrega en el orden de visita aplicando el algoritmo **depth first search**. Como punto de partida, debemos considerar el primer punto de entrega que fue añadido al mapa en su inicialización.
- Implementa un método, **rutaMinima**(), que reciba dos puntos de entrega, start y end, y que obtenga el camino mínimo desde start hasta end (aplicando

el algoritmo de Dijkstra). El método debe **devolver una lista de Python** con los puntos de entrega que componen dicho camino mínimo (incluyendo start y end), y además debe devolver la distancia entre start y end.

Se recomienda implementar la clase Mapa utilizando la implementación basada en diccionarios. En esta fase no se trabajaran ni con paquetes, zonas o repartidores. Sólo se trabajará con puntos de entrega.

En todas las fases, todas los métodos deben tener la menor complejidad temporal posible.

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.