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 2020-2021**	uc3m | Universidad Carlos III de Madrid

Campaña Vacunación Covid-19

Fase 1 - Listas

El objetivo de esta fase es permitir obtener información sobre el estado de la campaña de vacunación del Covid-19 en los centros de salud.

Cada centro de salud (clase HealthCenter) se define por un nombre (por ejemplo, "Los Frailes") y la lista de sus pacientes.

Nuestro sistema de salud guarda una información detallada de cada paciente, pero para este caso práctico, vamos a suponer que de cada paciente sólo es necesario conocer la siguiente información (ver clase Patient):

- name: Apellidos y Nombre (por ejemplo, "Segura, Isabel").
- year: Su año de nacimiento (por ejemplo, 1974)
- covid: Si ha pasado la covid (True para indicar si ha pasado la covid y False en otro caso).
- vaccine: Si ha sido vacunado o no (0 para indicar que no ha sido vacunado, 1 para indicar que ha recibido la primera dosis y 2 para indicar que ya ha recibido la segunda dosis).

Te proporcionamos la implementación de la clase Patient y parte de la implementación de la clase HealthCenter, que deberás modificar para incluir la siguiente funcionalidad:

- El método, **addPatient**, que reciba un paciente y lo añada a la lista de pacientes del centro de salud. La lista debe estar ordenada alfabéticamente y el método deberá insertar el nuevo paciente en su posición correspondiente.

El paciente sólo se añade si no está almacenado en la lista de pacientes. **La complejidad del método debería ser lineal.**

- El método, **searchPatients**, que reciba los siguientes parámetros:
    - year: el método buscará todos los pacientes nacidos en dicho año y en años anteriores. Ten en cuenta que si el parámetro recibe el año actual (2021), el método buscará todos los pacientes.
    - covid: Su valor por defecto será None, que indicará que se buscarán todos los pacientes, hayan o no sufrido la covid. Si el valor del argumento es True, el método deberá buscar los pacientes que han sufrido la covid. Si su valor es False, se buscarán los pacientes que no han sufrido la covid.
    - vaccine: Su valor por defecto será None, que indica que se buscarán todos los pacientes, hayan recibido alguna dosis o no. Si el valor del argumento es 0, el método deberá buscar los pacientes que no hayan recibido ninguna dosis. Si el valor es 1, el método deberá buscar los pacientes que sólo hayan recibido una dosis. Si el valor es 2, el método deberá buscar los pacientes que hayan recibido dos dosis.

  El método devolverá un nuevo centro cuya lista de pacientes cumpla los criterios de búsqueda definidos por los argumentos de entrada del método. **La complejidad del método debe ser lineal.**

- El método, **statistics**, que devuelva los siguientes valores:
    - Porcentaje de pacientes del centro de salud que ya han pasado la covid. El formato del porcentaje deberá ser 0.## (sólo dos decimales)
    - Porcentaje de pacientes mayores de 70 años (nacidos en 1950 o antes) que ya han pasado la covid. El formato del porcentaje deberá ser 0.## (sólo dos decimales)
    - Porcentaje de pacientes que aún quedan por vacunar (no han recibido ninguna dosis de la vacuna). El formato del porcentaje deberá ser 0.## (sólo dos decimales)
    - Porcentaje de mayores de 70 años (nacidos en 1950 o antes) que aún quedan por vacunar (no han recibido ninguna dosis de la vacuna). El formato del porcentaje deberá ser 0.## (sólo dos decimales)
    - Porcentaje de pacientes que ya han recibido la primera dosis. El formato del porcentaje deberá ser 0.## (sólo dos decimales)
    - Porcentaje de pacientes que ya han recibido la segunda dosis. El formato del porcentaje deberá ser 0.## (sólo dos decimales)

La complejidad del método statistics debe ser lineal:

- Un método, **merge**, que reciba un objeto de la clase HealthCenter, other, y que devuelva un nuevo centro de salud cuya lista de pacientes incluya a los pacientes del centro invocante, y también a los pacientes del centro other. Recuerda que la lista de pacientes debe estar ordenada de forma alfabética y

que no admite duplicados. En caso de duplicados, el nuevo centro sólo guardará el paciente del centro invocante (self). No está permitido que utilices un algoritmo de ordenación para ordenar dicha lista. **La complejidad del método debe ser lineal.**

- El método, **minus**, que reciba un objeto de la clase HealthCenter, other, y que devuelva un nuevo centro de salud que contenga a los pacientes del centro invocante, pero estos pacientes no pueden pertenecer al centro other. Recuerda que la lista de pacientes debe estar ordenada alfabéticamente, y que no admitir duplicados. No está permitido que utilices un algoritmo de ordenación para ordenar dicha lista. **Tu solución debe ser lo más eficiente posible.**
- Un método, **inter**, que reciba un objeto de la clase HealthCenter, other, y que devuelva un nuevo centro de salud cuya lista de pacientes sólo incluya a aquellos pacientes que pertenecen a ambos centros de salud. Recuerda que su lista de pacientes debe estar ordenada alfabéticamente, y que no admite duplicados. Es decir, cada paciente, aunque aparezca en los dos centros, sólo se añadirá una vez al nuevo centro. No está permitido que utilices un algoritmo de ordenación para ordenar dicha lista. **Tu solución debe ser lo más eficiente posible.**

Fase 2 - Árboles

El objetivo de esta fase es desarrollar una aplicación que permita gestionar la citación de los pacientes a vacunar en las próximas semanas.

En esta fase, seguiremos utilizando la clase *Patient* (definida en la fase 1), que permite representar un paciente y la información relevante para el COVID. Vamos a suponer que el nombre (apellido, nombre) es único para cada paciente. Además, en lugar de utilizar una lista enlazada para representar los pacientes de un centro de salud, se utilizará una estructura de datos, *HealthCenter2*, donde los pacientes están ordenados alfabéticamente (salvo que se diga lo contrario) y las operaciones de búsqueda, inserción y borrado tienen complejidad logarítmica.

Te proporcionamos la implementación de la clase *Patient* y una parte de la implementación de la clase *HealthCenter2*, que permite representar los pacientes de un centro de salud. Esta clase tiene la siguiente funcionalidad que deberás implementar

- El método, **searchPatients**, que reciba los siguientes parámetros:
    - *year*: el método buscará todos los pacientes nacidos en dicho año y en años anteriores. Ten en cuenta que si el parámetro recibe el año actual (2021), el método buscará todos los pacientes.
    - covid: Su valor por defecto será None, que indicará que se buscarán todos los pacientes, hayan o no sufrido la covid. Si el valor del argumento es True, el método deberá buscar los pacientes que han sufrido la covid. Si su valor es False, se buscarán los pacientes que no han sufrido la covid.
    - vaccine: Su valor por defecto será None, que indica que se buscarán todos los pacientes, hayan recibido alguna dosis o no. Si el valor del argumento es 0, el método deberá buscar los pacientes que no hayan recibido ninguna dosis. Si el valor es 1, el método deberá buscar los pacientes que sólo hayan recibido una dosis. Si el valor es 2, el método deberá buscar los pacientes que hayan recibido dos dosis.

  El método debe visitar a todos los pacientes para comprobar si cumple o no las condiciones especificadas por los argumentos de la función. Para acceder a los pacientes, este método deberá aplicar un **recorrido por niveles**. No está permitido utilizar otro tipo de recorridos. El método devuelve un nuevo objeto de la clase *HealthCenter2* con los pacientes que cumplen las condiciones definidas por los argumentos de la función. En este centro de salud, los pacientes deben estar ordenados alfabéticamente.

- El método, ***vaccine***, que reciba los siguientes argumentos:

    - *nombre* (apellido, nombre) del paciente a vacunar.
    - *vaccinated*, objeto de tipo *HealthCenter2*, donde los pacientes están almacenados de forma alfabética.

  Para simular la vacunación del paciente, el método deberá contemplar los siguientes casos:

  1) Si el paciente a vacunar no existe en el centro de salud *invocante*, el método muestra un mensaje informando que el paciente no existe y devuelve False.
  2) Si el paciente a vacunar existe en el centro de salud *invocante*, y ya había recibido las dos dosis correspondientes, el método deberá mostrar un mensaje informando que ese paciente ya había sido vacunado con anterioridad. Además, eliminará al paciente del centro de salud *invocante* y lo almacenará en el centro de salud *vaccinated*. La función devuelve False.
  3) Si el paciente a vacunar existe en el centro de salud *invocante*, y únicamente había recibido una dosis de la vacuna, el método actualizará su número de dosis a dos, eliminará al paciente del centro invocante. Por último, el paciente debe ser registrado en el centro de salud *vaccinated*. El método devuelve True.
  4) Si el paciente a vacunar existe en el centro de salud *invocante*, y no ha recibido ninguna dosis, el método simplemente deberá actualizar el número de dosis administradas al paciente. El método devuelve True.

- El método **makeAppointment** asignará una cita a un paciente. Este método recibe los siguientes argumentos:

    - *name*: el nombre (apellidos, nombre) del paciente a citar.

    - *appointment*: un string en formato “hh:mm” (por ejemplo, 08:00, 08:05, 09:55, 14:00, 18:30, 19:55, etc). Para simplificar el problema, suponemos que los minutos de la cita debe ser 00 o un número múltiplo de 5. Los horarios de vacunación son desde las 08:00 a las 19:55 h. Para simplificar el problema, suponemos que todas las citas son para el mismo día.

    - *schedule*: un objeto de tipo HealthCenter2, que contiene a pacientes que ya han sido citados. En este caso, los pacientes no están ordenados en base a su nombre (apellido, nombre), sino en base a su cita (string en formato hora:minutos). Como se ha dicho anteriormente, todas las citas son en el mismo día.

Para simular la asignación de una cita a un determinado paciente, el método *makeAppointment,* deberá contemplar los siguientes casos:

1) Si el paciente a citar no existe en el centro de salud invocante, la cita no se crea. El método muestra un mensaje informando que el paciente no existe y devuelve False.

2) Si el paciente existe en el centro invocante y ha recibido las dos dosis, la cita no se creará. El método informará que el paciente ya había sido vacunado y devuelve False.

3) Si el paciente existe en el centro invocante y no ha recibido las dos dosis, el método debe almacenar al paciente junto con su cita en el objeto *schedule*. Para añadir el paciente a schedule, se deberán contemplar los siguiente casos:
    - Si la cita (appointment) está libre, el paciente se añade a schedule. El método devuelve True.
    - Si la cita no está libre, la función deberá buscar el hueco disponible más próximo en el tiempo. Si existen dos huecos disponibles con la misma distancia en el tiempo (por ejemplo, 5 minutos antes y 5 minutos después de la cita deseada), se le asignará la cita más temprana en el tiempo (es decir, la de 5 minutos antes). Una vez encontrado la hora y minutos disponible para ser citado, el método debe añadir el paciente a schedule. El método devuelve True.
    - Si no hay citas disponibles, el método informará que no existen citas disponibles y devolverá False.

Nota: Se permite que un mismo paciente, tenga varias citas en un día.

- ¿Cuál es la complejidad temporal de cada función: searchPatients, vaccine y makeAppointment?. Razona el mejor y peor caso, indicando su complejidad. Puedes añadir tu respuesta en un comentario de la función.

Fase 3. Grafos

Cada día, la Consejería de Salud de la Comunidad de Madrid recibe un **mapa** (clase Map) con los distintos centros de salud donde se deberán entregar vacunas. Dicho mapa está compuesto por los centros de salud (vértices). Para simplificar el caso práctico, en esta fase únicamente almacenamos el nombre de cada centro de salud (es decir, no vamos a almacenar sus pacientes).
El mapa, además de los centros de salud, también representa las posibles conexiones o caminos entre los centros de salud. Así, si dos centros de salud están directamente conectados, el mapa contendrá la distancia en km entre ambos centros de salud. En el mapa, existen centros de salud que no están directamente conectados, por lo que será necesario pasar por uno o varios centros de salud intermedios.
En esta fase, ya te proporcionamos la implementación de la estructura de datos Map, con la siguiente funcionalidad:

- El método **addHealthCenter**() que permite añadir un nuevo centro de salud al mapa.
- El método **addConection**() que recibe dos centros de salud y su distancia en km. El método crea una conexión entre ambos centros.
- El método **removeConnection**() que recibe dos centros de salud y elimina su conexión.
- El método **__str__**() que muestra todos los centros de salud y sus conexiones directas con otros centros y sus distancias.
- El método **areConnected**() que compruebe si dos centros de salud están **directamente conectados**. Si están conectados devuelve la distancia, en otro caso devuelve 0.
- El método **createPath**() que permite visitar todos los centros de salud. El método devuelve una lista de Python que contiene los centros de salud en el orden de visita aplicando el algoritmo **depth first search**. Como centro de partida, se considera el primer centro de salud que fue añadido al mapa en su inicialización.
- El método, **minimumPath**(), que reciba dos centros de salud, start y end, y que obtiene el camino mínimo desde start hasta end (aplicando el algoritmo de Dijkstra). El método devuelve una lista de Python con los puntos de entrega que componen dicho camino mínimo (incluyendo start y end), y además devuelve la distancia de start a end.

Además, del algoritmo del camino de Dijkstra, existen otros algoritmos otros algoritmos para obtener el camino mínimo (menor distancia en km) entre dos vértices de un grafos, en concreto:

- El algoritmo de Bellman-Ford
- El algoritmo de Floyd-Warshall

El algoritmo de **Bellman-Ford** calcula y devuelve el camino mínimo con su coste asociado. Si encuentra una conexión negativa devuelve falso. Al igual que en algoritmo de Dijkstra, se utiliza en array, d, para almacenar las distancias desde un vértice origen, s, a cada uno de los vértices del grafo. Así, para cada vértice v del grafo, d[v] almacena el coste del camino mínimo desde el vértice origen s hasta v. También se utiliza una array, p, para almacenar los vértices anteriores a cada vértice en el camino desde el vértice origen. A continuación, se muestra el pseudocódigo de este algoritmo:

```
Bellman-Ford (G,start)

Inicializar
    para cada vértice v
        hacer d[v] = infinito
              p[v] = nulo
    d[start] = 0

para 1 a |V|-1:
     para cada vértice u:
         para cada conexión (u,v)
             Si d[v] > d[u] + w(u,v) entonces
                 d[v] = d[u] + w(u,v)
                 p(v) = u

Para cada conexión (u,v) comprobar
    Si d[v] > d[u] + w(u,v) entonces
        devuelve FALSO 'el algoritmo no converge


devuelve d, p
```

El algoritmo de **Floyd-Warshall** permite encontrar el camino mínimo entre todos los posibles pares de vértices de un grafo. Utiliza la matriz de adyacencias (con los pesos de cada conexión) y una matriz P para almacenar los caminos mínimos. A continuación, se muestra el pseudocódigo de este algoritmo:

```
Floyd-Warshall (G)

Inicializar

    Sea D la matriz de adyacencias (distancias entre vértices)

    Sea P una matriz para almacenar los caminos mínimos



Si i=j o Dij= infinito entonces Pi,j = nulo

sino Pi,j=i
```

```
Para k = 0 hasta len(V)-1
        Para i = 0 hasta len(V)-1
            Para j = 0 hasta len(V)-1
                    Si Di,j > Di,k + Dk,j entonces
                       Di,j = Di,k + Dk,j
                        Pi,j = Pk,j
```

En esta tercera fase, también se pide:

- Implementar el método **minimumPath,** descrito anteriormente, aplicando el algoritmo de Bellman-Ford.
- Implementar el método **minimumPath,** descrito anteriormente, aplicando el algoritmo de Floyd-Warshall.

En la implementación de todas las 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.