Facultad de Ingeniería
UNER

Trabajo Práctico N°2:
Aplicaciones de Estructuras Jerárquicas y Grafos

Asignatura: Algoritmos y Estructuras de Datos

2° cuatrimestre - Año 2023

Fecha de entrega: 23/10/2023

Integrantes:

- **Borghi, Julián**
- **Claure Endara, Jorge**
- **Diaz Pinel, Enzo**

Ejercicio 1

El **triaje** es un proceso que permite una gestión del riesgo clínico para poder manejar adecuadamente y con seguridad los flujos de pacientes cuando la demanda y las necesidades clínicas superan a los recursos. El siguiente proyecto de software posee una sencilla simulación de esta situación: enlace a código.

En este proyecto para simplificar existen pacientes con tres niveles de riesgo para su salud: 1: crítico, 2: moderado, 3: bajo. Actualmente la simulación muestra que los pacientes se atienden según el orden en que llegan al centro de salud.

Se solicita:

a. Seleccionar, programar y aplicar una estructura de datos adecuada para almacenar los pacientes conforme ingresan al centro de salud **de modo tal que cuando se atiende un paciente siempre sea aquel cuyo nivel de riesgo es el más delicado** en comparación con el resto de los pacientes que restan por ser atendidos. Si dos pacientes poseen el mismo nivel de riesgo, adoptar un segundo criterio para seleccionar uno de ellos. Recordar que la estructura de datos debe ser genérica, es decir, debe poder almacenar cualquier tipo de dato, y no ser específica para alojar pacientes (separar implementación de aplicación).

b. Fundamentar en el informe, en no más de una página, la estructura seleccionada indicando el orden de complejidad O de inserciones y de eliminaciones en la estructura seleccionada.

   La estructura de datos que se utilizó para cumplir con la consigna fue un montículo ya que cumple con la comparación entre pacientes a partir de su complejidad y luego cumplir con el orden en el que serán atendidos. Esto se debe a que, a la hora que un paciente se atiende (es decir, se quita la raíz del montículo) se realiza una comparación entre la complejidad de los pacientes (que serían los valores que forman parte de cada hoja) y se coloca el que sea menor como primero (a la hora de determinar la nueva raíz), particularmente en este caso donde utilizamos un montículo mínimo.

   El montículo es una estructura de datos que se puede esquematizar como un árbol, pero para su almacenamiento se puede usar una lista. Se utiliza para resolver este problema ya que los montículos poseen eficiencia en las operaciones de inserción y eliminación para empezar. Por un lado, la inserción de un elemento tiene una complejidad de O(log n), donde 'n' es el número de elementos en el montículo. Esto se debe a la necesidad de ajustar el montículo después de agregar un nuevo elemento. Por otro lado, la eliminación de un elemento mínimo se realiza en O(log n) también. Esto implica la reorganización del montículo para mantener su estructura de montículo válida.

Observación: Pueden realizarse todas las modificaciones al código que crean necesarias siempre que se respete el propósito del problema planteado.

Ejercicio 2

Kevin Kelvin es un científico que estudia el clima y, como parte de su investigación, debe consultar frecuentemente en una base de datos la temperatura del planeta tierra dentro de un rango de fechas. A su vez, el conjunto de medidas crece conforme el científico registra y agrega **mediciones** a la base de datos.

Una medición está conformada por el valor de temperatura en ºC (flotante) y la fecha de registro, la cual deberá ser ingresada como “dd/mm/aaaa” (string). (Internamente sugerimos emplear objetos de tipo *datetime*).

Ayude a Kevin a realizar sus consultas eficientemente implementando una base de datos en memoria principal “**Temperaturas_DB**” que utilice internamente un árbol AVL. La base de datos debe permitir realizar las siguientes operaciones (interfaz de Temperaturas_DB):

guardar_temperatura(temperatura, fecha): guarda la medida de temperatura asociada a la fecha.

devolver_temperatura(fecha): devuelve la medida de temperatura en la fecha determinada.

max_temp_rango(fecha1, fecha2): devuelve la temperatura máxima entre los rangos fecha1 y fecha2 inclusive (fecha1 < fecha2). Esto no implica que los intervalos del rango deban ser fechas incluidas previamente en el árbol.

min_temp_rango(fecha1, fecha2): devuelve la temperatura mínima entre los rangos fecha1 y fecha2 inclusive (fecha1 < fecha2). Esto no implica que los intervalos del rango deban ser fechas incluidas previamente en el árbol.

temp_extremos_rango(fecha1, fecha2): devuelve la temperatura mínima y máxima entre los rangos fecha1 y fecha2 inclusive (fecha1 < fecha2).

borrar_temperatura(fecha): recibe una fecha y elimina del árbol la medición correspondiente a esa fecha.

devolver_temperaturas(fecha1, fecha2): devuelve un listado de las mediciones de temperatura en el rango recibido por parámetro con el formato “dd/mm/aaaa: temperatura ºC”, ordenado por fechas.

cantidad_muestras(): devuelve la cantidad de muestras de la BD.

Adicionalmente realizar las siguientes actividades:

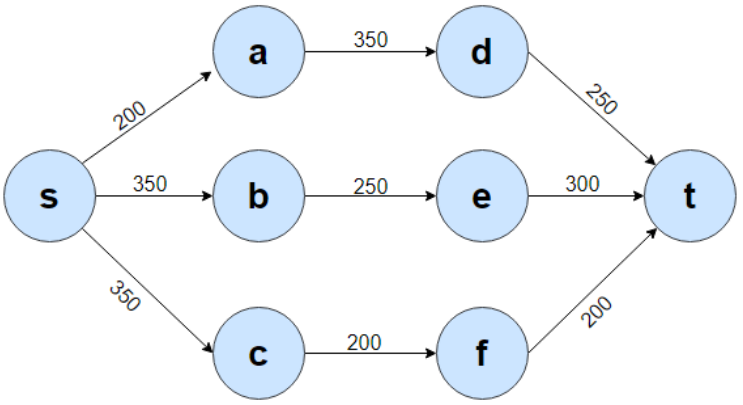
- Escriba una tabla con el análisis del orden de complejidad Big-O para cada uno de los métodos implementados para su clase “Temperaturas_DB”, explique brevemente el análisis de los mismos.

Ejercicio 3

La empresa CasaBella S.A. en Ciudad de Buenos Aires fabrica y vende mobiliario de hogares en distintas ciudades y ha decidido tercerizar la distribución de sus productos. Para esto, contrata un servicio de transporte para llevar sus productos de una ciudad a otra. El responsable de contabilidad de la empresa tiene un archivo con una lista de todas las rutas disponibles, cada ruta en la lista es una tupla (s_i, t_i, w_i, c_i) donde s_i y t_i son respectivamente los nombres (string) de la ciudad de inicio y final de cada ruta de transporte, w_i es un entero positivo que representa la capacidad máxima de peso admitida (en kg) en esa ruta y c_i un entero positivo que representa el precio (en unidades de 1000) de realizar el transporte por la misma (el precio es el mismo si el camión se traslada vacío o a su máxima capacidad). La existencia de una ruta de s_i a t_i no implica la existencia de la ruta t_i a s_i a menos que esté indicado explícitamente.

La capacidad máxima de peso admitida (w_i) en una ruta va a estar limitada por el tramo que tenga menor capacidad de peso en esa ruta, a este valor se lo denomina “cuello de botella”.

Este tipo de problema es conocido como “El problema del camino más amplio o del máximo cuello de botella”. Por ejemplo, en el siguiente grafo tenemos 3 rutas (o caminos) entre los vértices “s” y “t” y los pesos en las aristas representan capacidades de peso.



El cuello de botella del primer camino (superior) es 200, porque es el peso de la arista con menor capacidad. Los cuellos de botella de los caminos restantes son 250 y 200 (medio e inferior respectivamente). El máximo cuello de botella del grafo es 250 y será por tanto la máxima capacidad que se podrá transportar entre los vértices “s” y “t”. Alternativamente, si se encontraran dos rutas con el mismo cuello máximo, entonces, se tendrán dos posibles caminos para transportar la misma cantidad de productos.

Ayude a la empresa CasaBella a evaluar sus opciones de transporte implementando un algoritmo que devuelva:

1. El peso máximo w_{max} que se pueda transportar desde la ciudad de Buenos Aires a cualquier otra ciudad de destino.En este ítem, hallar el peso máximo w_{max} que se

pueda transportar desde la ciudad de Buenos Aires a una ciudad de destino equivale a encontrar el máximo cuello de botella entre ambas ciudades.

2. El precio mínimo para transportar un mobiliario del peso w_{max} desde la ciudad de Buenos Aires a otra ciudad de destino.

Utilice el archivo provisto en el siguiente enlace para representar el problema planteado y encontrar la solución a los ítems anteriores.