PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

ESTRUCTURA DE DATOS Y PROGRAMACIÓN METÓDICA

Examen 1
(Segundo Semestre 2023)

Duración: 2h 50 min.

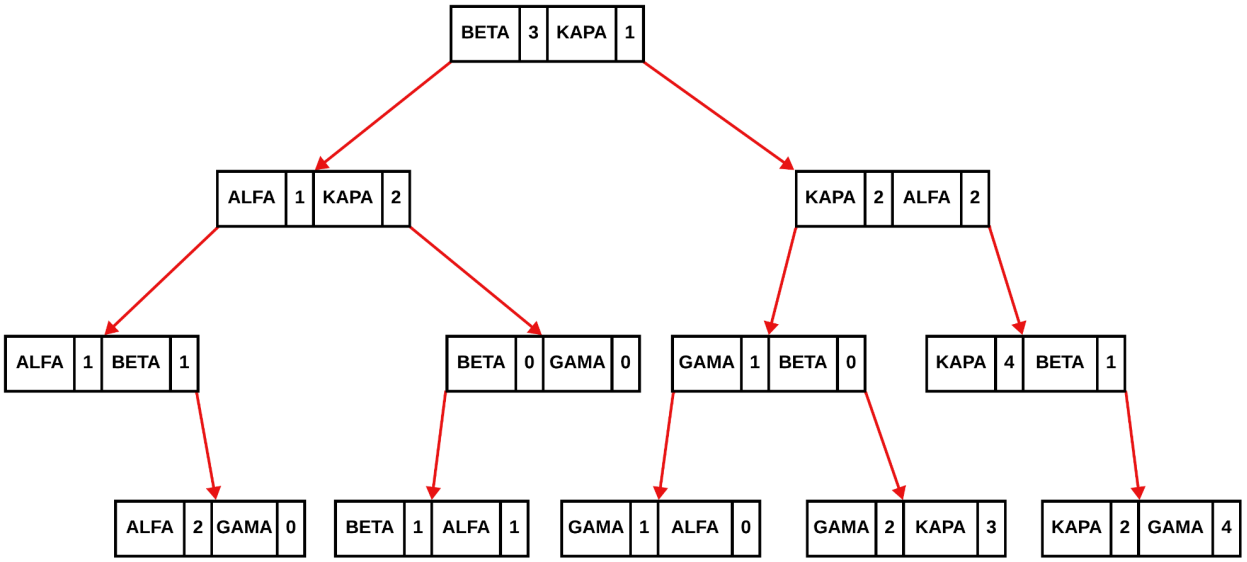
- Los programas deben ser desarrollados en el lenguaje C++ sobre Netbeans, cualquier otro IDE no será evaluado.
- Un programa que no muestre resultados coherentes y/o útiles será corregido sobre el 50% del puntaje asignado a dicha pregunta. Si el programa no compila o tiene errores tendrá descuentos significativos en la nota final.
- Debe utilizar comentarios para explicar la lógica seguida en el programa elaborado.
- El orden será parte de la evaluación.
- Se utilizarán herramientas para la detección de plagios, por tal motivo si se encuentran soluciones similares, se anulará la evaluación a todos los implicados y se procederá con las medidas disciplinarias dispuestas por la FCI.
- Su trabajo deberá ser subido a PAIDEIA.
- Se permite usar el material desarrollado en clase, pero no compartirlo.
- Los archivos deben tener el siguiente formato: **E1_codigo_PX.zip**, donde X es el número de la pregunta.

PREGUNTA 1 (10 puntos)

Se ha realizado un campeonato de fútbol en el que han participado los equipos ALFA, BETA, GAMA Y KAPA. Los resultados de los partidos están almacenados en el archivo de texto llamado **campeonato.txt**. (está disponible en Paideia para ser descargado con el enunciado del examen). En cada línea de dicho archivo hay cuatro datos de un partido: el nombre del equipo local, la cantidad de goles anotados por el equipo local, el nombre del equipo visitante y la cantidad de goles anotados por el equipo visitante.

Los datos contenidos en el archivo serán cargados en un árbol binario de búsqueda. **La clave o criterio de ordenamiento de este árbol será la combinación (concatenación) del nombre del equipo local y el nombre del equipo visitante**.

En la siguiente figura se muestra el árbol árbol generado con los datos del archivo:



Se declaran las siguientes estructuras para ser usadas en la elaboración del programa:

```
struct Partido
{
      char local[15];          // nombre del equipo local
      int golesLocal;          // goles anotados por el equipo local
      char visitante[15];      // nombre del equipo visitante
      int golesVisitante;      // goles anotados por el equipo visitante
};

struct NodoCampeonato
{
      Partido partido;              // datos del partido
      struct NodoCampeonato *hizq;  // puntero al hijo izquierdo
      struct NodoCampeonato *hder;  // puntero al hijo derecho
};

struct ArbolCampeonato
{
      NodoCampeonato * raiz;        // puntero a la raíz del árbol
};

struct NodoPuntaje
{
      char equipo[15];          // nombre del equipo
      int puntaje;              // puntaje obtenido en el campeonato
      struct NodoPuntaje *sig; // puntero al siguiente nodo
};
```

Implemente las siguientes funciones, dónde arbolBB es el árbol binario de búsqueda:

Una función **cargarPartidos** que permita construir un árbol binario de búsqueda para almacenar la información del campeonato en base a los partidos que están registrados en el archivo de texto ***campeonato.txt***. *strcat(cadena1, cadena2), concatena cadena1 con cadena2 y lo devuelve en cadena1.* *Parámetros:* **nombre del equipo** y otros que sean necesarios para implementar la función.	2 puntos
Una función llamada **mostrarPartidosEquipo** que permite mostrar los resultados de todos los partidos en los que ha participado un equipo. *Parámetros:* **nombre del equipo** y otros que sean necesarios para implementar la función.	2 puntos
Una función llamada **obtenerEstadisticaEquipo** que devuelva para un equipo los siguientes datos: cantidad de partidos ganados, la cantidad de partidos empatados, la cantidad de partidos perdidos, la cantidad de goles a favor, la cantidad de goles en contra y el puntaje total obtenido en el campeonato. Tenga en cuenta que un partido ganado vale 3 puntos, un partido empatado vale un punto y un partido perdido vale cero puntos.	3 puntos

Parámetros: **nombre del equipo** y otros que sean necesarios para implementar la función.	
Una función llamada **elaborarTablaPosiciones** que permita generar una lista enlazada simple que contenga en cada uno de sus nodos el nombre de un equipo y el puntaje que obtuvo en el campeonato. Esta lista debe estar ordenada por puntaje de mayor a menor. Si hay dos o más equipos con el mismo puntaje pueden aparecer en cualquier orden. Debe también mostrar el contenido de esta lista. *Parámetros:* **nombre del equipo** y otros que sean necesarios para implementar la función.	3 puntos

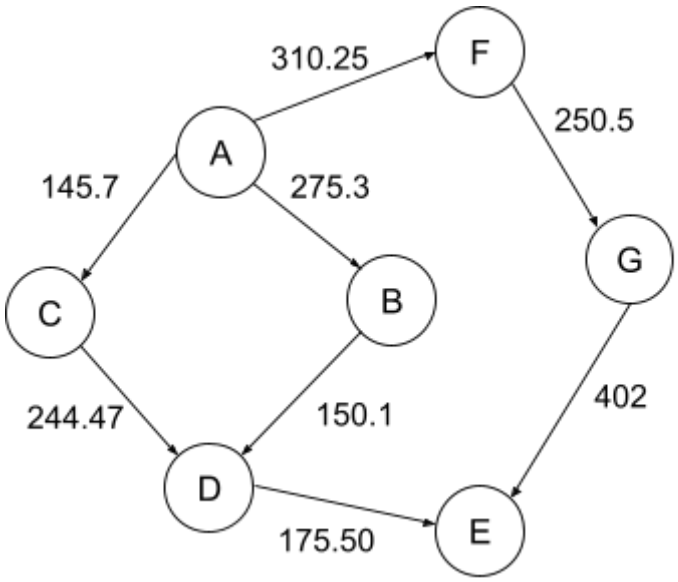
Debe desarrollar también la función **main()** que permita probar las funciones.

Nota: Se permite implementar otras funciones que ayuden en la solución, pero solo se califican las mencionadas anteriormente. No es posible agregar otros campos en las estructuras mencionadas al comienzo del enunciado de la pregunta.

PREGUNTA 2 (10 puntos)

Supongamos que tienes un mapa de ciudades como el que se muestra a continuación.

Figura 1.



Donde las ciudades están representadas por las letras A, B, C, D, E, F, y G. Solo hay ciertas conexiones entre ciudades. Por ejemplo, de la ciudad A puede ir a la ciudad C, B y F, de la ciudad B puede ir a la ciudad D, de E no se puede ir a ninguna ciudad. Las conexiones entre ciudades y la distancia que hay entre ciudades se dan en la Figura 1.

Se le pide **crear un grafo ponderado que representa este mapa de ciudades con las distancias proporcionadas**.

Para ello debe implementar las siguientes funciones:

(2 puntos) agregarVertice(grafo, letraCiudadOrigen). Función que agrega al grafo una letra que representa el nombre de **una ciudad.**

```
struct NodoListaVertice{
        char letraCiudadOrigen;          // letra que representa a la ciudad
        struct NodoListaVertice * siguiente;
        struct ListaArista listaArista;
};
```

(3 puntos) agregarArista(grafo, letraCiudadOrigen, letraCiudadDestino, distancia). Función que agrega una arista entre el vértice origen y el vértice destino, recibe como último parámetro la **distancia** entre ciudades. Realice los cambios necesarios a su código para que al crear el Nodo de la Lista de Aristas pueda guardarse esta información.

```
struct Ciudad
{
        char letraCiudadDestino;        // letra que representa a la ciudad
        double distancia;               // distancia para llegar a la ciudad
};

struct NodoListaArista
{
        struct Ciudad ciudad;           // ciudad que se conecta con el vértice
        struct NodoListaArista * siguiente;
};
```

En el programa principal llame a la función que le muestre el grafo y las aristas para comprobar que lo ha construido correctamente. Se tomará en cuenta la impresión en la corrección. Un ejemplo de salida es el siguiente. Utilice el formato que le parezca más adecuado.

A: F, 310.25 C, 145.7 B, 275.3

B: D, 150.1

C: D, 244.47

D: E, 175.5

E:

F: G, 250.5

G: E, 250.5

(2 puntos) dameVecinosDeUnVertice(grafo, letraCiudadOrigen, pila). Función que *recorre el grafo buscando una ciudad y apila sus ciudades vecinas en la variable pila.* Para esta función necesita construir una **pila** y que el elemento de la pilas sea un **struct Ciudad.** Esto quiere decir que la pila de **apilar** un *struct Ciudad* y **desapila** un *struct Ciudad.* Muestre los resultados en el programa principal.

```
struct NodoPila
{
        struct Ciudad ciudad;           // ciudad que se conecta con el vértice
        struct NodoPila * siguiente;
};
```

(1 punto) distanciaMinimaEntreUnVerticeYVecinos(grafo, pila, letraCiudadOrigen, minDistacia, letraCiudadMinDistancia). Función que calcula la distancia mínima entre una ciudad y sus ciudades vecinas. Y, devuelva en los parámetros "**minDistacia**", la mínima distancia entre ellas y en el parámetro "**letraCiudadMinDistancia**", la letra correspondiente a la ciudad con la mínima distancia. Recibe en uno de sus parámetros la variable **pila** (llena) con las ciudades vecinas a la ciudad origen. Muestre los resultados en el programa principal.

(2 puntos) mostrarTodosRecorridosDesdeUnaCiudad(grafo, letraCiudadOrigen). Función que muestra todos los caminos posibles desde una ciudad hasta donde termina cada uno de sus recorridos en el grafo. Es decir todas las trayectorias posibles desde una ciudad hasta donde se pueda llegar. Por ejemplo, como 'A' tiene tres ciudades vecinas, tendremos tres trayectorias. Para el desarrollo de esta función puede hacer uso del siguiente algoritmo.

El algoritmo hace uso de una **pila** para ir guardando las ciudades vecinas y las que va visitando.

Apilar todas las ciudades vecinas a la ciudad origen

Mientras la pila no esté vacía

    desapila una ciudad vecina (V)

    Si existe conexión entre las ciudades

        imprime ciudad origen y la ciudad vecina (V)

    caso contrario

        imprime la ciudad vecina (V)

    Busca en el grafo la ciudad vecina (V)

        Si la encuentro, apilo sus ciudades vecinas

Fin Mientras

El resultado usando 'A' como ciudad origen, debe ser el siguiente:

A: B D E

A: C D E

A: F G E

Profesores del curso: ANA RONCAL DE GUANIRA
HINOJOSA LAZO, HILMAR ANTONIO

San Miguel, 9 de octubre del 2023