LABORATORIO IV

GONZALO DE VARONA

A00358687

ALGORTIMOS Y PROGRAMACIÓN 2

UNIVERSIDAD ICESI

OCTUBRE 19 DE 2019

**Objetivos**

**Unidad 4: Estructuras y Algoritmos Recursivos**

OE4.1 Emplear el concepto de recursividad como una alternativa a la estructura de control iterativa.

OE4.2 Aplicar la computación recursiva en la solución de problemas de naturaleza inherentemente autocontenida.

OE4.3 Utilizar árboles binarios de búsqueda para representar grupos de objetos que mantienen entre ellos una relación de orden.

OE4.4 Escribir algoritmos recursivos para manipular estructuras de información recursivas y explicar las ventajas que, en este caso, estos algoritmos tienen sobre los algoritmos iterativos.

**Enunciado**

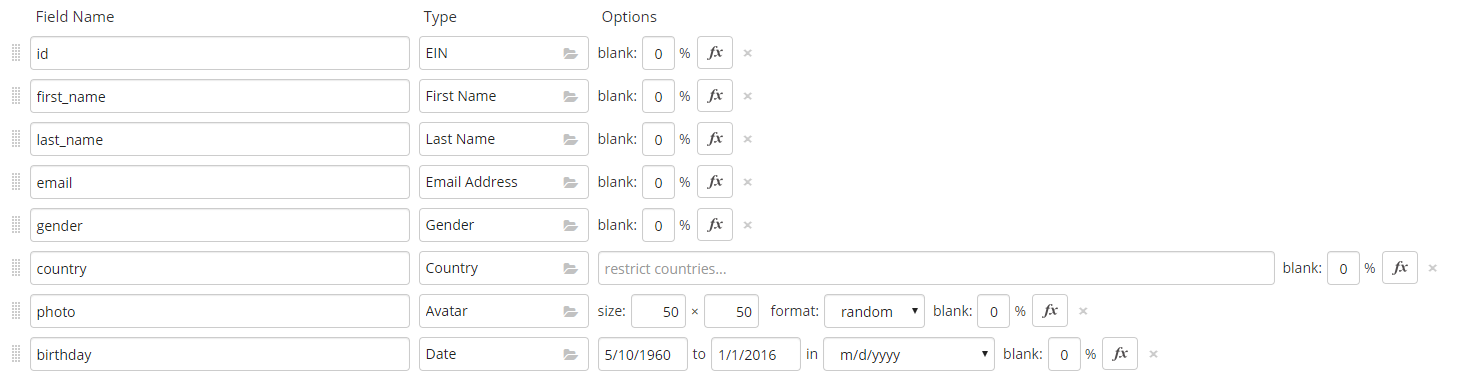
La [IV Copa Panamericana de Voleibol Masculino Sub-21](https://es.wikipedia.org/wiki/Copa_Panamericana_de_Voleibol_Masculino_Sub-21_de_2019) se celebrará del 5 al 11 de mayo de 2019 en Lima, (Perú) y la federación tiene la necesidad de registrar algunos datos de los espectadores que desean asistir al evento y por motivo de algunas labores de logística durante el evento, el personal de registro y control estará permanentemente realizando búsquedas de espectadores y participantes en el software, utilizando como criterio el id de registro del evento.

Usted ha sido elegido para desarrollar un programa que permita leer la información de los posibles asistentes al evento desde un archivo de texto delimitado por comas y cargar la información de manera que se puedan hacer búsquedas a través del identificador del registro del evento.

El programa debe permitir leer el archivo de texto con la información de todos los posibles participantes. Esta información se cargará en una estructura de datos de tipo árbol binario de búsqueda (ABB) cuya relación de orden estará dada por el id de registro del evento. Una vez cargada esta información en el árbol de espectadores, se seleccionará aleatoriamente el 50% de esos espectadores para agregarlos a una lista doblemente enlazada, que será la estructura de datos que almacenará los inscritos al evento o participantes al evento.

Luego de la lectura del archivo y la creación de las dos estructuras de datos: una para los posibles participantes y otra para los inscritos al evento, se debe permitir hacer búsquedas por id, para cada una de las estructuras creadas. Su programa debe indicar, en cualquier caso, si el espectador se encuentra o no entre los buscados así como el tiempo que tardó la búsqueda. Si el espectador fue encontrado, deben visualizarse sus datos en un panel a la izquierda.

Los datos de entrada al programa pueden ser generados utilizando la herramienta web [**https://www.mockaroo.com**](https://www.mockaroo.com/)que permite generar datos aleatorios de forma personalizada. La configuración y orden de los datos generados debe ser la que se muestra a continuación:



Se debe generar el máximo de filas posible para hacer buenas pruebas y el formato en que se debe exportar debe ser CSV el cual delimita los archivos de texto con coma por defecto.

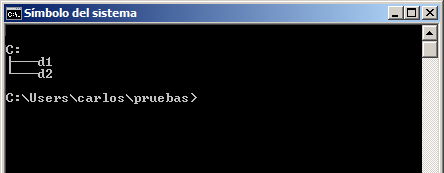


Ya que el máximo es 1000, realice pruebas del tiempo de la consulta para archivos de entrada mas grande como 100000, por ejemplo (generando 100 veces 1000 y uniéndolos). Compare los tiempos de las búsquedas cuando se realizan sobre todos los programadores y los participantes.

La interfaz del programa debe cumplir con los lineamientos:

* Permite cargar los datos de un archivo plano (ejemplo: archivo.csv) y confirmar la correcta carga, la ruta que se entra por teclado puede ser relativa o absoluta, debe especificarle al usuario final cual debe ingresar.
* Buscar espectador por id, debe cuantificar cuánto tiempo tarda la búsqueda y mostrar los datos del espectador (la foto en esta versión no será visualizada).
* Buscar participantes por id, debe cuantificar cuánto tiempo tarda la búsqueda y mostrar los datos del participante (la foto en esta versión no será visualizada).

El programa también contará con una opción que podrá visualizar los datos de las dos estructuras de un país determinado, donde la estructura del árbol binario de búsqueda se verá como un árbol y la lista se visualizará en forma consecutiva. La estructura árbol puede ser vista de la siguiente manera:



Donde c: es la raíz del árbol y d1 y d2 son hijos de c:,para cada opción se visualizará sólo los datos del país seleccionado (puede ser el componente JOptionPane de Java), eso quiere decir, que para la estructura de espectadores el árbol pintado solo debe tener espectadores de la ciudad seleccionada e igual para la estructura de los participantes.

**Entregables.** **1.** Requerimientos. **2.** Diagrama de clases de modelo e interfaz (no generado automáticamente). **3.** Implementación completa de todos los requerimientos en Java. **4.** Diseño de casos de pruebas unitarias de todas las funcionalidades relacionadas con los árboles. **5.** Implementación de todas las pruebas unitarias diseñadas y corriendo perfectamente. **6**. Todos los métodos para administrar las estructuras deben ser implementados usando recursividad. **7**. La estructura de los espectadores deben ser un árboles binarios de búsqueda y los participantes una lista doblemente enlazada, tal como se explicaron en el tema visto en clase y no podrán usar otro tipo de contenedor como array, ArrayList, etc.

**Fecha de Entrega:** viernes 19 de octubre de 2019 a las 11:59 p.m. a través de Moodle.

**Integrantes**: el laboratorio debe realizarse individualmente.

|  |  |
| --- | --- |
| **Nombre** | **R.F. 1. Buscar un espectador** |
| **Resumen** | El usuario debe ingresar el nombre del espectador, para buscarlo en el sistema. |
| **Entradas** | |
| -ID del espectador | |
| **Resultados** | |
| Espectador encontrado. | |

|  |  |
| --- | --- |
| **Nombre** | **R.F. 2. Buscar un participante** |
| **Resumen** | El usuario debe ingresar el nombre del participante, para buscarlo en el sistema. |
| **Entradas** | |
| -ID del participante | |
| **Resultados** | |
| Participante encontrado. | |

|  |  |
| --- | --- |
| **Nombre** | **R.F. 3. Cargar archivos para el sistema** |
| **Resumen** | El usuario debe ingresar la ruta donde se encuentra el archivo csv, para cargar los datos en el sistema. |
| **Entradas** | |
| -Ruta del archivo | |
| **Resultados** | |
| Archivo cargado. | |

|  |  |
| --- | --- |
| **Nombre** | **R.F. 4. Mostrar tiempos de busqueda** |
| **Resumen** | El programa debe de mostrar los tiempos de busqueda de un espectador o de un participante |
| **Entradas** | |
| <ninguna> | |
| **Resultados** | |
| Tiempos de búsqueda mostrados. | |

|  |  |
| --- | --- |
| **Nombre** | **R.F. 5. Mostrar espectadores de un país en forma de árbol binario** |
| **Resumen** | El usuario debe ingresar el país de los espectadores, los cuales se mostraran en pantalla en forma de árbol binario. |
| **Entradas** | |
| -Nombre del país de los espectadores | |
| **Resultados** | |
| Espectadores de un país mostrados como un árbol binario. | |

|  |  |
| --- | --- |
| **Nombre** | **R.F. 5. Mostrar participantes de un país** |
| **Resumen** | El usuario debe ingresar el país de los participantes, los cuales se mostraran en pantalla en forma de una lista. |
| **Entradas** | |
| -Nombre del país de los participantes | |
| **Resultados** | |
| Participantes de un país mostrados como una lista. | |

**Diseño de casos de pruebas unitarias**

**Manager:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Valor Entrada** | **Resultado** |
| Manager | addPlayer(Player)  searchPlayer(String) | El escenario inicializa un manager (coordinador) el cual no tiene ningún jugador asociado | 2 | Verdadero, porque al realizar la búsqueda, se encuentra que el resultado es un String con el mensaje de error por defecto si no encuentra al jugador requerido. |
| Manager | addPlayer(Player)  searchPlayer(String) | El escenario inicializa un manager (coordinador) el cual tiene 4 jugadores asociados | 2 | Verdadero, porque al realizar la búsqueda, se encuentra que en la cadena del resultado contiene el nombre del jugador asociado al ID = 2, por consiguiente encontró y agregó correctamente al jugador |

**Objetivo:** Probar si el método de agregar y buscar por ID algún jugador funciona correctamente

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Valor Entrada** | **Resultado** |
| Manager | searchPlayersByCountry() | El escenario inicializa un manager (coordinador) el cual no tiene ningún jugador asociado | USA | Verdadero, porque al realizar la búsqueda, se encuentra que el resultado es un String que no contiene ni siquiera un nombre de los jugadores de USA |
| Manager | searchPlayersByCountry() | El escenario inicializa un manager (coordinador) el cual tiene 4 jugadores asociados | USA | Verdadero, porque al realizar la búsqueda, se encuentra que el resultado es un String que contiene todos los nombres de los jugadores de USA |

**Objetivo:** Probar si se encuentran los jugadores provenientes de algún país con el fin de imprimirlos por pantalla.

**Spectator:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Valores entrada** | **Resultado** |
| Spectator | searchSpectator(String) | El escenario inicializa un árbol abb balanceado con peso 3 | 1 | Verdadero, ya que se encuentra el espectador esperado |
| Spectator | searchSpectator(String) | El escenario inicializa un árbol abb no balanceado con peso 2 donde no hay ramificación izquierda | 3 | Verdadero, ya que se encuentra el espectador esperado |

**Objetivo:** Probar si el método está creando correctamente las minas y las casillas libres.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Valores entrada** | **Resultado** |
| Spectator | searchSpectator(String) | El escenario inicializa un árbol abb no balanceado con peso 2 donde no hay ramificación derecha | 1 | Verdadero, ya que se encuentra el espectador esperado |
| Spectator | searchSpectator(String) | El escenario inicializa un árbol abb balanceado con peso 1 | 2 | Verdadero, ya que se encuentra el espectador esperado. |

**Objetivo:** Probar si el método de buscar un espectador por ID funcionan correctamente.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Valores entrada** | **Resultado** |
| Spectator | addSpectator (Spectator) | El escenario inicializa un árbol abb balanceado con peso 3 | 6 | Verdadero, ya que se encuentra el espectador esperado |
| Spectator | addSpectator (Spectator) | El escenario inicializa un árbol abb no balanceado con peso 2 donde no hay ramificación izquierda | 6 | Verdadero, ya que se encuentra el espectador esperado |

**Objetivo:** Probar si el método modifica el atributo valor para cualquier casilla

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Valores Entrada** | **Resultado** |
| Spectator | addSpectator (Spectator) | El escenario inicializa un árbol abb no balanceado con peso 2 donde no hay ramificación derecha | 6 | Verdadero, ya que se encuentra el espectador esperado |
| Spectator | addSpectator (Spectator) | El escenario inicializa un árbol abb balanceado con peso 1 | 6 | Verdadero, ya que se encuentra el espectador esperado |

**Objetivo:** Probar si el método agrega correctamente un espectador nuevo en el árbol binario de búsqueda.