DESARROLLO DE ALGORITMOS

Informe de Sistema para Gestionar Campeonatos de Fútbol

Huilcaleo Malena [FAI 4704]
Grupo N° 10
30/06/2024

Introducción

Este informe detalla las estructuras datos, así como la comparación de métodos de ordenamiento “divide y venceras” y “por fuerza bruta” y función recursiva utilizadas en el desarrollo de un sistema de gestión de campeonatos de futbol.

Objetivos del Proyecto

Registro de equipos y jugadores

- Importar archivos de jugadores y equipos para agilizar la carga de datos.
- Cada equipo tiene un nombre, categoría, datos asociados al torneo y un conjunto de jugadores.
- Cada jugador tiene nombre, apellido, edad, DNI y numero de camiseta.

Armado de Fixture

- Generar automáticamente el calendario de fechas asegurando que no se repitan enfrentamientos entre equipos.
- Para una cantidad de equipos impar asignar una fecha libre para cada equipo.

Ingreso de resultados

- Registrar resultados de cada fecha una única vez.
- Verificar que los números de camiseta correspondan a un jugador del equipo.

Gestión de Jugadores

- Permitir incorporar nuevos jugadores verificando que no existan previamente en algún equipo y que el numero de camiseta no esté en uso dentro del equipo.

Consulta de tablas

- Mostrar en cualquier momento la tabla de posiciones del torneo ordenada por puntos, y a igual cantidad de puntos, por diferencia de goles en forma descendente.
- Ver resultados de una fecha específica que haya sido cargada.
- Ver tabla de goleadores del torneo.

Consulta de estadísticas

- Edad promedio de todos los jugadores calculada de manera recursiva.
- Contar la cantidad de jugadores cuya edad supera en 5 a la edad promedio de todos los jugadores.
- Encontrar el primer jugador de un equipo que tenga menos que una edad dada de manera recursiva.

Visualizar lista de jugadores

- Mostrar lista ordenada de jugadores por apellido y nombre usando un método de ordenamiento por fuerza bruta y otro por divide y vencerás.

Estructuras de Datos Utilizadas

Tipos de Datos Abstractos(TDA)

Clase Jugador

La clase Jugador gestiona y mantiene la información individual de los jugadores que participan del torneo

Atributos

- nombre, apellido: Representan el nombre y apellido del jugador, útil para el ordenamiento de jugadores.
- Edad: Edad del jugador, necesaria para calcular estadísticas de los jugadores como edad promedio.
- dni: Indica el DNI identificando de manera única al jugador.
- numCamiseta: Numero de camiseta del jugador, único dentro de cada equipo.
- cantGoles: Cantidad de goles que marco el jugador, no suman goles que haya hecho en contra.
- nomEquipo: Referencia al equipo que pertenece el jugador.

Constructores

Jugador(Texto ape, Texto nom, Entero docu, Entero ed, Entero num, Texto nomEq): Este contructor solicita todos los atributos necesarios para inicializar un objeto de la clase, excepto la cantidad de goles(cantGoles), ya que un jugador siempre comienza con cero goles al momento de ser creado y este valor se modifica según avanza la competencia.

Jugador(*int docu*): Este constructor solo solicita el dni que es el identificador único, edad, numCamiseta y cantidad se inicia en cero, nomEquipo, nombre y apellido como cadenas vacias

Metodos Destacados

- añadirGol(): Este método permite registrar un gol, incrementando el contador interno de goles marcados del jugador(cantGoles).
- equals(Jugador otroJugador): Compara el jugador actual con otro jugador usando el dni ya que es el atributo de identificación única del jugador.
- compareTo(Jugador otroJugador): Compara dos jugadores según la cantidad de goles que realizo cada uno y devuelve un valor entero negativo, positivo o cero dependendiendo si un jugador realizo más, menos o igual cantidad de goles.
- compareToNombres(Jugador otroJugador): Este método compara dos jugadores según su apellido, siguiendo un orden alfabetico. Si los apellidos son iguales, se compara por nombre. Retorna un valor negativo, positivo, o cero según si el jugador actual es menor, igual o mayor en términos de apellidos y nombres.

UML Jugador

Jugador
<->Cadena nombre <->Cadena apellido <->Entero edad <->Entero dni <->Entero numCamiseta <->Entero cantGoles <->Cadena nomEquipo

//Constructores
Jugador(Texto ape, Texto nom, Entero docu, Entero ed, Entero num, Texto nomEq)
Jugador(Entero docu)
 	
//Modificadores
<+>setNombre(): void
<+>setApellido():void
<+>añadirGol():void
<+>setEdad(Entero ed):void
 	
//Observadores
<+>getNombre(): Texto
<+>getApellido (): Texto
<+>getNomEquipo(): Texto
<+>getCamiseta (): Entero
<+>getCantGoles (): Entero
<+>getDni (): Entero
<+>getEdad (): Entero
 	
//Propias del tipo
<+>compareTo(Jugador otroJugador): Entero
<+>compareToNombres(Jugador otroJugador): Entero
<+>equals(Equipo otroEquipo): Logico

Clase Equipo

Esta clase encapsula los atributos y métodos necesarios para maneja la representación de cada equipo que participa en el torneo y los datos asociados a este.

Atributos

- nombre: Representa el nombre del equipo, identificando de manera única a cada equipo participante del torneo.
- categoria: Indica la categoria actual del equipo pudiendo variar desde ‘A’ a la ‘F’
- ganados, perdidos, empatados, jugados: Contadores que permiten llevar un registro del rendimiento del equipo.
- golesaFavor, golesEnContra: Representan el total de goles marcados a favor y en contra respectivamente.
- Jugadores: Arreglo que almacena la lista de los jugadores que conforman el equipo
- cantJugadores: Contador que lleva registro de la cantidad actual de jugadores que componen el equipo

Constructor

Equipo(String nombre, char categoria): Inicializa un nuevo equipo con el nombre y la categoría especificados. También inicializa los contadores en 0.

La elección de no tener constructores con parametros formales de partidos jugados, ganados, perdidos, empatados, goles a favor, goles en contra se debe a que esto refleja la situación inicial que tiene un partido antes de comenzar el torneo, es decir, ningún equipo comienza la competencia con un puntaje o partidos jugados.

Métodos Destacados

- actualizarEstadisticas(Entero aFavor, Entero enContra): Permite actualizar las estadísticas del equipo después de cada partido ajustando los contadores ganados, perdidos y empatados y actualiza los goles a favor y en contra.
- puntosAcumulados(): Calcula el total de puntos que tiene el equipo al momento de su llamado usando los atributos ganados y empatados. Este método es fundamental en la determinación de la posición de cada equipo en la tabla de posiciones.
- diferenciaGoles(): Usa los atributos golesaFavor y golesEnContra para calcular la diferencia de goles que además es otro factor importante para la tabla de posiciones.
- compareTo(Equipo otroEquipo): Compara dos equipos por puntos acumulados y, en caso de igual de puntos, por diferencia de goles. Esto es importante define como se llevará a cabo el ordenamiento de la tabla de posiciones del torneo.
- buscarJugadorPorDni(Entero dni): Este método busca a un jugador por dni en el arreglo de jugadores, es útil para no cargar dos veces un mismo jugador.
- buscarJUgadorPorCamiseta(Entero numCamiseta): De similar manera al método anterior, busca a un jugador, pero esta vez por número de camiseta. Es necesario para evitar que haya jugadores con el mismo número de camiseta dentro del equipo.
- toString(): Utilizado para la visualización de la tabla de posiciones ya que retorna una cadena con el nombre del equipo y las estadísticas.

UML Equipo

Equipo
<->Cadena nombre <-> Caracter categoría <->Entero jugados <->Entero ganados <->Entero perdidos <->Entero empatados <->Entero golesaFavor <->Entero golesEnContra <->Entero cantJugadores <-> Jugador [ ] jugadores

//Constructores
Equipo(Texto nombre, Caracter categoria)
 	
//Modificadores
<+>aumentarCategoria(): void
<+>disminuirCategoria():void
<+>actualizarEstadisticas(Entero aFavor, Entero enContra):void
<+>agregarJugador(Jugador nuevoJugador):void

```
//Observadores
<+>getNombre(): Texto
<+>getJugadores(): Jugador[ ]
<+>getCategoria(): Caracter
<+>getJugados(): Entero
<+>getGanados (): Entero
<+>getPerdidos(): Entero
<+>getEmpatados(): Entero
<+>getGolesaFavor(): Entero
<+>getGolesEnContra(): Entero
<+>getCantJugadores(): Entero
<+>toString(): Texto
<+>mostrarJugadoresEquipo(): Texto

//Propias del tipo
<+>puntosAcumulados(): Entero
<+>diferenciaGoles(): Entero
<+>compareTo(Equipo otroEquipo): Entero
<+>buscarJugadorPorCamiseta(Entero numCamiseta): Jugador
<+>buscarJugadorPorDni(Entero dni): Jugador
<+>equals(Equipo otroEquipo): Logico
```

Clase Partido

La clase partida representa un encuentro entre dos equipo, gestionando los goles marcados por cada equipo y la carga de resultados.

Atributos

- equipoLocal, equipoVisitantes: Objetos del tipo ‘Equipo’ que representan los equipos que se enfretaron en el encuentro.
- golesvisitante, golesLocales: Enteros que representan la cantidad de goles marcados de cada equipo.
- cargaResultados: Logico que indican si los resultados han sido o no cargados, fundamental para que una fecha no sea cargada mas de una vez.

Constructor

Partido(Equipo local, Equipo Visitante): Inicializa un partido entre el equipo local y un equipo visitante. Los goles son inicializados en cero y la carga de resultados en falso.

La elección de realizar un solo constructor se debe a que cada partido es creado durante el armado del fixture, es decir, previo a que el partido sea jugado por lo que aun no hay un resultados del encuentro y en consecuencia la carga no puede realizarse inmediatamente.

Metodos Relevantes

- esPartidoDescanso (): Verifica si el partido es un descanso, retornando ‘true’ si alguno de los equipos se llama “Descanso”
- cargarResultados(Entero goles, Entero golesV): Permite cargar los resultados del partido estableciendo la cantidad de goles marcados por el equipo local y visitante. Ademas actualiza los goles de los equipos involucrados.

- setCargaResultados(): Me permite modificar la cargaResultados cuando uno de los equipos se llama “Descanso”
- toStringFechas(): Retorna una cadena en un formato para ver el fixture, es decir no indica la cantidades de goles.
- toStringResultados(): Similar al anterior pero este si da los resultados del partido.

Concepto de 'Descanso'

El término 'Descanso‘ se refiere a la situación en la que uno de los dos equipos no participa activamente en una fecha específica, lo que implica que no acumula goles ni estadísticas de partido. Esta circunstancia se da cuando la cantidad de equipos que participan del torneo es impar, asignando a cada equipo una fecha de descanso o libre en la cual no participará. Esto se representa mediante un equipo llamado 'Descanso'.

UML Partido

Partido
<-> Equipo equipoLocal <-> Equipo equipoVisitante <->Entero golesLocales <->Entero golesVisitantes <->Logico cargaResultados

//Constructores
Partido(Equipo local, Equipo Visitante)
 	
//Modificadores
<+>cargarResultados (entero golesL, entero golesV): void
<+>setCargaResultados ():void
 	
//Observadores
<+>getGolesLocales (): Entero
<+>getGolesVisitantes (): Entero
<+>getEquipoVisitante (): Equipo
<+>getEquipoLocal(): Equipo
<+>toStringFechas():Texto
<+>toStringResultados(): Texto
 	
//Propias del tipo
<+> esPartidoDescanso ():Logico

Arreglos

Los arreglos juegan un papel fundamental en este proyecto para la manipulación y organización de los datos de los equipos y jugadores. A continuación, áreas donde los arreglos son utilizados:

Listas de equipos:

Los equipos participantes del torneo se almacenan en un arreglo dentro de la clase Campeonato donde cada equipo se representa como un objeto de la clase ‘Equipo’.
Funcionalidades del arreglo de Equipos

- Lista de posiciones: Utilizando este arreglo, se puede generar y mantener actualizada la tabla de posiciones del torneo.
- Cargar los jugadores en Equipos: El arreglo de equipos facilita la asignación de jugadores a sus respectivos equipos.
- Generación de fixture: El arreglo de equipos también se utiliza para generar el fixture del torneo y se asegurase que cada equipo juegue contra todos los demás equipos participantes.
- Determinación de descanso: De acuerdo a la cantidad de equipos cargados en este arreglo se determina si es necesario asignar un descanso en cada fecha. Esto es de acuerdo a si la cantidad de equipos almacenados en este arreglo es par o impar.
- Al agregar un nuevo jugador: Cuando se quiere agregar un nuevo jugador se debe verificar que no pertenezca a ninguno de los equipos que está almacenado en este arreglo.

Lista de Jugadores por equipos:

Cada equipo posee un arreglo del tipo 'Jugador ', donde se almacenan los jugadores que forman parte de ese equipo. Este arreglo permite:

- Verificar Disponibilidad de Camisetas: Controlar que una camiseta no esté siendo utilizada por otro jugador dentro del equipo.
- Búsqueda de Jugador por Edad: Facilita la búsqueda de un jugador dentro del equipo que tenga menos de una edad determinada.

Lista de todos los jugadores del torneo:

La combinación de los arreglos anteriores permite construir un arreglo con todos los jugadores que participan en el torneo. Esto proporciona un acceso rápido a funcionalidades como:

- Calculo de edad promedio: Permite calcular rápidamente la edad promedio de todos los jugadores.
- Verificación de jugadores únicos: Al agregar un nuevo jugador, se verifica que no esté registrado.
- Ordenar todos los jugadores por Apellido y Nombre
- Ordenar todos los jugadores según cantidad de goles

Matrices

En este proyecto se emplea una matriz de tipo Partido para gestionar el fixture del torneo. Esta matriz se dimensiona con base en dos parámetros clave:

1. Cantidad de fechas(cantFechas): Determina la cantidad de filas en la matriz. Cada fila representa una fecha del torneo donde se juegan varios partidos.
   Se calcula tomando la longitud del arreglo arrTemp (una versión temporal de los equipos participantes no sobredimensionado y con longitud par), menos uno. Esto asegura que haya suficientes fechas para que todos los equipos se enfrenten al menos una vez.
2. Cantidad de partidos(partidosFecha): Define la cantidad de columnas en la matriz. Cada columna entones representa un partido.
   Se calcula dividiendo la longitud de arrTemp entre 2. Esto distribuye equitativamente los partidos entre las fechas, garantizando que cada equipo juegue exactamente una vez por fecha.

Funcionalidades de la matriz:

- Armado de fixture: se organiza y almacena los partidos de cada fecha.
- Carga de los partidos: a través de la matriz [] [] fechas se puede cargar los resultados de partidos de cada fecha, la carga se realiza una única vez por fecha(fila).
- Visualización de las fechas ya jugadas: Se utiliza la matriz para mostrar los resultados de una fecha en específico, pero no sin antes verificar que haya sido cargada. Esto implica recorrer la matriz y verificar si los partidos han sido cargados.

Método de ordenamientos

En este proyecto se usaron dos métodos para la funcionalidad ”Ver Jugadores”. Se permite al usuario elegir entre un método por fuerza bruta y un método de divide y venceras

Método de ordenamiento por Fuerza Bruta

El método de ordenamiento por fuerza bruta elegido es burbuja mejorado, que a diferencia de burbuja reduce el número de comparaciones cuando el arreglo ya está parcialmente ordenado.

Ventajas:

- Fácil de implementar.
- Adecuado para listas pequeñas o listas que ya están casi ordenadas.

Desventajas:

- No es eficiente para listas grandes debido a su complejidad de tiempo O(n^2)

Codigo de implementación:

```java
public static void ordenarJugadoresSegunNombre(Jugador[]arr){
        Jugador aux;
        int n=arr.length,i=0,j;
        boolean ordenado=false;

        while(i< n-1 && !ordenado){
            ordenado=true;
            for(j=0; j<n-i-1; j++){
                if(arr[j].compareToNombres(arr[j+1])>0){
                    ordenado= false;
                    aux=arr[j];
                    arr[j]=arr[j+1];
                    arr[j+1]=aux;
                }

            }
            i++;
        }

    }
```

Método de ordenamiento divide y venceras

El método de ordenamiento divide y venceras elegido es MergeSort.

Ventajas:

- Eficiente para arreglos grandes.
- Complejidad O(nlogn).

Desventajas:

- Requiere espacio adicional para almacenar las sublistas temporales.
- Más complejo de implementar.

Código de implementación:

```java
public static void mergeSortPorNombre(Jugador[] arr, int ini, int fin) {
        if (ini < fin) {
            int mitad = (ini + fin) / 2;
            mergeSortPorNombre(arr, ini, mitad);
            mergeSortPorNombre(arr, mitad + 1, fin);
            merge(arr, ini, mitad, fin);
        }
    }
    public static void merge(Jugador[] arr, int ini, int mitad, int fin) {
        int posA = ini, posB = mitad + 1, posC = 0;
        Jugador[] aux = new Jugador[fin - ini + 1];

        while (posA <= mitad && posB <= fin) {
            if (arr[posA].compareToNombres(arr[posB]) < 0) {
                aux[posC++] = arr[posA++];
            } else {
                aux[posC++] = arr[posB++];
            }
        }

        while (posA <= mitad) {
            aux[posC++] = arr[posA++];
        }

        while (posB <= fin) {
            aux[posC++] = arr[posB++];
        }
        posC=0;
        while (posC<aux.length) {
            arr[ini + posC] = aux[posC];
            posC++;
        }
    }
```

El método “mergeSortPorNombre” divide recursivamente el arreglo en mitades hasta llegar a subarreglos individuales. Luego, utiliza la función “merge” para combinar y ordenar estos subarreglos de manera que los jugadores queden ordenados por sus nombres alfabéticamente.

Pruebas empíricas

Se realizaron pruebas empíricas para evaluar el rendimiento de los métodos de ordenamientos mencionados anteriormente.

Para realizar las pruebas se utilizaron diferentes tamaños de arreglos de tipo Jugador con tamaños que van desde pequeños hasta grandes, para cada tamaño cada algoritmo se ejecutó varias veces para capturar el tiempo promedio de medición. Estas mediciones fueron realizadas utilizando ‘System.nanoTime()’.

Resultados

Tamaño del Array	Burbuja Mejorada (ns)	Merge Sort (ns)
15	98.900	55.800
30	349.700	108.000
60	899.800	223.900
80	1.490.100	349.700
100	1.785.500	440.700
150	2.264.500	665.900
200	3.045.800	856.200
250	3.870.800	1.026.500
300	5.116.700	1.153.700

Conclusiones de las pruebas:

De acuerdo a las pruebas empíricas realizadas MergeSort resulta significativamente superior a Burbuja mejorado, sobre todo en arreglos grandes. Mientras que MergeSort mantiene tiempos razonables a medida que crece el tamaño del arreglo Burbuja mejorada aumenta considerablemente el tiempo requerido para ordenar los elementos.

Función recursiva

La función 'cantJugadoresSuperanProm' cuenta cuantos jugadores tienen una edad que supera en 5 al promedio de todos los jugadores que participan del torneo

Caso base:

El caso base esta dado de manera implícita, este ocurre cuando i es igual o mayor a la longitud del arreglo o cuando el elemento del arreglo en la posición i es nulo.

Caso Recursivo:

Cuando el jugador en la posición i del arreglo arr tiene una edad mayor que edadProm + 5, se incrementa el contador cont en 1 y se llama recursivamente a la función con i + 1.

Si el jugador no supera esa edad, simplemente se llama recursivamente a la función con i + 1.

Retorno: Finalmente, el valor de cont, que se acumula recursivamente sumando 1 por cada jugador que supera la edad promedio más 5 años, es devuelto como resultado total de la función.

Implementación del código:

```java
public static int cantJugadoresSuperanProm(int i, int edadProm, Jugador[]arr){
        int cont=0;
        if(i<arr.length && arr[i]!=null){
            Jugador temp= arr[i];
            if(temp.getEdad()>(edadProm+5)){
                cont=1+cantJugadoresSuperanProm(i+1, edadProm, arr);
            }else{
                cont=cantJugadoresSuperanProm(i+1, edadProm, arr);
            }
        }
        return cont;
    }
```