Algoritmos y Estructuras de Datos I

PROYECTO INDIVIDUAL – PARTE I

<GONZALO, ADROVER>

Contenido

Introducción

Este trabajo se realizó con el objetivo de contestar las preguntas indicadas de la forma más optima posible. Para ello, se utilizaron diversas estructuras en el análisis de datos para poder recabar la información necesaria.

El proyecto final es capaz no solo de contestar las preguntas, sino que también lo logra hacer en un tiempo considerablemente menor comparado con otras alternativas. A su vez, también fue desarrollado de la manera más clara posible, cumpliendo con las convenciones más populares.

Problema planteado

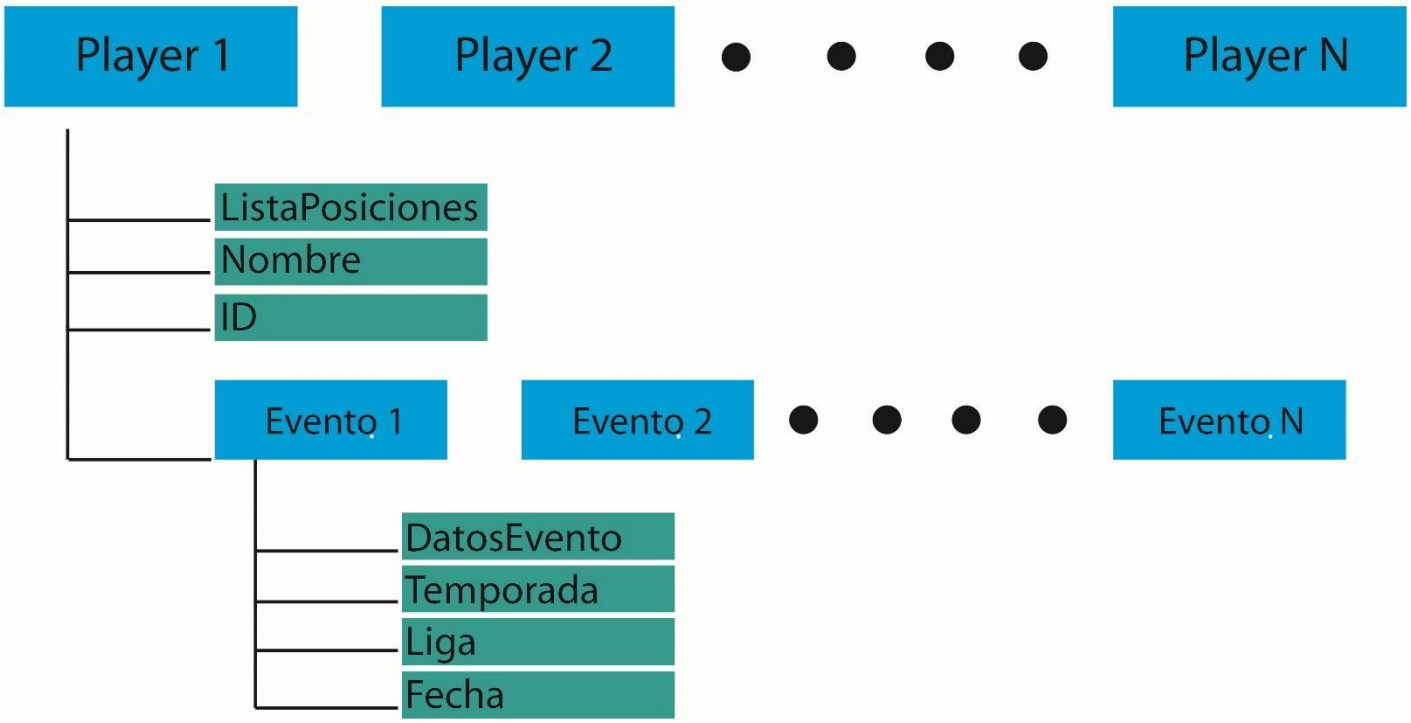
El problema inicial se trata de que, teniendo una base de datos de importante volumen, se deberán procesar los mismos para poder contestar interrogantes planteadas, siendo el eje del problema encontrar el once ideal.

Para lograr un buen resultado se debe no solo llegar a las respuestas sino también pensar alternativas para elegir entre ellas la más eficiente. También parte del problema es encontrar de que manera cargar los datos para que se pueda acceder a estos de una manera efectiva.

Análisis de alternativas

Se consideraron dos alternativas basadas en listas TDA. A continuación, se explicarán las mismas junto con diagramas ilustrativos.

Alternativa 1:



Esta es la estructura utilizada actualmente. Esta estructura trata de una lista de jugadores, donde cada uno tiene cuatro atributos: Nombre, ID, lista de posiciones y lista de eventos. A su vez la lista de eventos en los que participó cuenta con datos del evento, temporada, liga y una fecha dada.

Costo de memoria:

N° de referencias a jugadores en lista = N = Cantidad de jugadores.

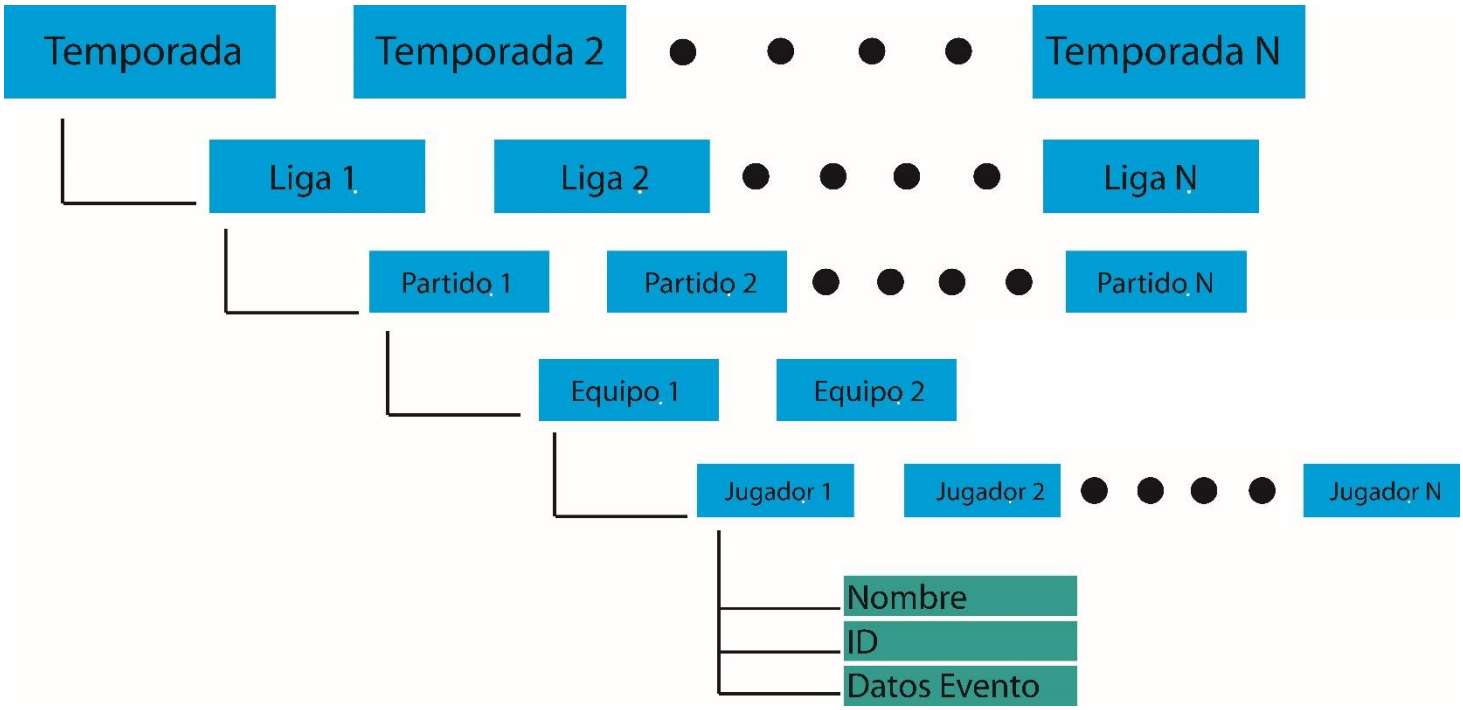
N° de referencias por jugador = 4.

N° de referencias de eventos = Q = Cantidad de eventos.

N° de referencias por evento = 4.

N° total de referencias en estructura = 4*N + 4*Q = 4 * (N+Q)

Alternativa 2:



En esta estructura opcional se pensó de forma piramidal, como se organizarían los datos en la vida real. Aquí cuenta con una lista de temporadas, cada temporada tiene una lista de ligas, cada liga tiene una lista de partidos, cada partido tiene una lista de equipos (2), y cada equipo tiene una lista de jugadores, donde cada jugador tiene tres atributos: Nombre, ID y Datos de eventos. Se terminó descartando por el costo de memoria y el orden de ejecución; además de su complejidad a la hora de analizar como se almacenarían los datos.

Costo de memoria:

N° de referencias a Temporadas = N = Cantidad de Temporadas.

N° de referencias por Temporadas = N * Q = Cantidad de Temporadas * Cantidad de Ligas.

N° de referencias de Ligas = Q = Cantidad de Ligas.

N° de referencias por ligas = Q * S = Cantidad de Ligas * Cantidad de Partidos.

N° de referencias de Partidos = S = Cantidad de Partidos.

N° de referencias por Partidos = S*2 (equipos).

N° de referencias de Jugadores = S * 22 (número de jugadores) * 3 (numero de datos por jugador).

N° total de referencias en estructura = N + N*Q + Q + Q*S + S*2 + S*66 = N + (S * 68) + (Q * (N + S + 1))

Algoritmos

Pregunta 1:

Alternativa 1:

O8

Pregunta1(Lista : ListaJugadores, Temporada : String) : String

nodoactual ← Lista.primero O(1)

listaPartidos = new ListaTupla O(1)

MIENTRAS (nodoactual <> nulo) HACER O(N)

nodoevento = actual.getDato.listaEventos.getPrimero O(1)

MIENTRAS (nodoevento <> nulo) HACER O(M)

SI < nodoevento.getDato.getTemporada = Temporada > ENTONCES O(1)

nodoPartido = listaPartidos.buscar(actual.getDato.jugadorID) O(1)

SI (nodoPartido <> null) HACER O(Ñ)

cantidadParticipaciones ← nodoPartido.getDato[1] O(1)

cantidadParticipaciones += 1 O(1)

listaPartidos.getDato ← cantidadParticipaciones O(1)

SINO

array ← {actual.getDato.jugadorID,1,actual.getDato.nombre,evento.getDato.getLiga} O(1)

nodo ← nuevo Nodo(nodoactual.getDato.jugadorID,array) O(1)

listaPArtidos.insertar(nodo) O(1)

FINSI

FINSI

Actualizar nodoevento O(1)

FIN MIENTRAS

Actualizar nodoactual O(1)

FIN MIENTRAS

vacio ← {nulo, 0, nulo, nulo} O(1)

actual2 ← listaPartidos.primero O(1)

MIENTRAS (actual2 <> nulo) HACER O(R)

SI (actual2.getDato[1] > vacio[1]) ENTONCES O(1)

vacio[0] = temporada.getDato()[0]; O(1)

vacio[1] = temporada.getDato()[1]; O(1)

vacio[2] = temporada.getDato()[2]; O(1)

vacio[3] = temporada.getDato()[3]; O(1)

FINSI

actualizar actual2 O(1)

FIN MIENTRAS

El algoritmo es de orden N (cantidad de jugadores), por la regla de la suma. En dicho algoritmo la sentencia dominante es aquella de la lista de los jugadores, los otros ordenes son inferiores. Fue seleccionado este algoritmo debido a su bajo orden y que en comparación con el de la alternativa este era más fácil de implementar y de menor tiempo de ejecución.

Pregunta 3:

Alternativa 1:

Pregunta3(Lista : ListaJugadores , Nombre : String) : String

nodoactual ← Lista.primero O(1)

ligas ← ListaString O(1)

MIENTRAS (nodoactual <> nulo) HACER O(N)

SI (Nombre = nodoactual.getDato.nombre) ENTONCES O(M)

nodoevento = nodoactual.getDato.listaEventos.getPrimero O(1)

MIENTRAS (nodoevento <> nulo) HACER O(R)

nodoLiga ← ligas.buscar(actual.getDato.getLeague) O(1)

SI (nodoLiga = nulo) ENTONCES O(1)

ligas.insertar(nodo ← nuevo Nodo(nodoevento.getDato.getLIga, nodoevento.getDato.getLIga) O(1)

FINSI

actualizar nodoevento O(1)

FIN MIENTRAS

FINSI

actualizar nodoactual O (1)

FIN MIENTRAS

DEVOLVER ligas.imprimirDato

El algoritmo es de orden N (cantidad de jugadores), debido a la regla de la suma. Aquí la sentencia dominante es la lista de los jugadores, mientras que las otras son de menor orden. Se selecciono este algoritmo debido a su bajo orden comparado con lo que hubiera sido la otra alternativa. En la otra se debería haber recorrido la lista de jugadores además de todas las demás generando un mayor tiempo de ejecución.

La estructura se repite para las otras preguntas por lo cual la implementación es análoga para el resto de preguntas.

Selección y justificación de alternativa a implementar

La elección final fue por la Alternativa 1. Principalmente esto se debe en que a la hora de contestar las preguntas, esta alternativa era más fácil de llevar a cabo, con algoritmos menos complejos, menos clases y con menor tiempo de ejecución. Además, se pudo hacer uso de los TDA lista para el desarrollo de este.

La clase Main es la encargada de vincular lo que sucede en Respuestas y en View. Aquí se instancia a Respuestas y a View, pasándole a la instancia cargar como parámetro. Finalmente se inicializa el Modelo de View que permite seleccionar respuestas. De esta manera se logra separar la interfaz de usuario con la clase View, de la encargada de realizar las operaciones lógicas, Respuestas.

En la primera ejecución se genera la estructura de datos dentro de la clase Respuestas. Para esto la clase llama métodos de LectorArchivos en un orden determinado.

Aquí se carga la lista de jugadores desde “players.csv” y luego se le asigna posiciones desde “match-standings.csv”. Despues se crea una pila de eventos desde “events.csv” y se carga la información de partidos desde “matches.csv”. Finalmente, a la pila de eventos se la actualiza con los datos de los partidos y también se cargan los eventos en los jugadores.

Para el caso de los eventos se decidió optar por el TDA Pila y no el TDA Lista ya que en primeros modelos se observó que con estos se tardaba mucho debido a su extenso proceso para agregar nuevos elementos. En la pila se realiza un push para cada evento agregando después del último pero sin recorrer toda la pila, como sucedería con el TDA lista.

Una vez cargados todos los datos, los métodos de Respuestas tienen la información necesaria para poder contestar lo que se les solicite.

Pregunta 1:

Método encargado de contestar la pregunta 1. En esta se solicita contestar cual fue el jugador que tuvo más participación en una temporada en una misma liga y en qué liga fue.

Para esto se crea un nodo con el primer jugador de la lista de jugadores y se itera. Para cada jugador se toma el primer evento de la lista de eventos y también se itera.

Si la temporada del evento coincide con la que indicó el usuario.

Si coincide chequea si existe un nodo partido para este jugador. Si no está registrado se le asigna una cierto array con su id, un 1, el nombre del jugador y su liga. Si el equipo ya tiene este array

creado se le suma uno a la cantidad de participaciones (pos 2 del array). Para finalizar se corrobora quien es el que tiene más y se retorna ese jugador junto con su liga.

Recibe por parámetro Temporada indicada por el usuario.

Pregunta 2:

Para esta pregunta por cuestiones de tiempo se logró contestar la parte b, el peor delantero.

Para esto se calcula quién fue el delantero que erró el mayor número de tiros en una determinada temporada.

Pregunta 3:

Para esta pregunta se solicita un nombre y se retorna todas las ligas donde participó un jugador.

Pregunta 4:

Método encargado de contestar la pregunta 4. En esta se solicita contestar cual fue el equipo que efectuó mas goles y el que recibió más.

Para esto se crea un nodo con el primer jugador de la lista de jugadores y se itera. Para cada jugador se toma el primer evento de la lista de eventos y también se itera.

Si la temporada del evento coincide con la que indicó el usuario, se chequea si ocurrió un gol.

De haber uno se crea un array de objects con los datos del equipo del evento y del oponente. Si el equipo no esta registrado se le asigna una cierto array con su id y dos espacios vacios donde irán los goles a favor y en contra. Si el equipo ya tiene este array creado se le suman goles a favor o en contra (dependiendo si es el que metió el gol o oponente).

Para finalizar los que tienen más goles a favor y en contra en variables diferentes y se retornan

Toma como parámetro la Temporada indicada por el usuario.

Pregunta 5:

Método encargado de contestar la pregunta principal, el once ideal. Para esto se tienen en cuenta los criterio de mejor delantero (mayor cantidad de goles), mejor mediocampista (mayor numero de asistencias) y mejor defensa (menor numero de faltas).

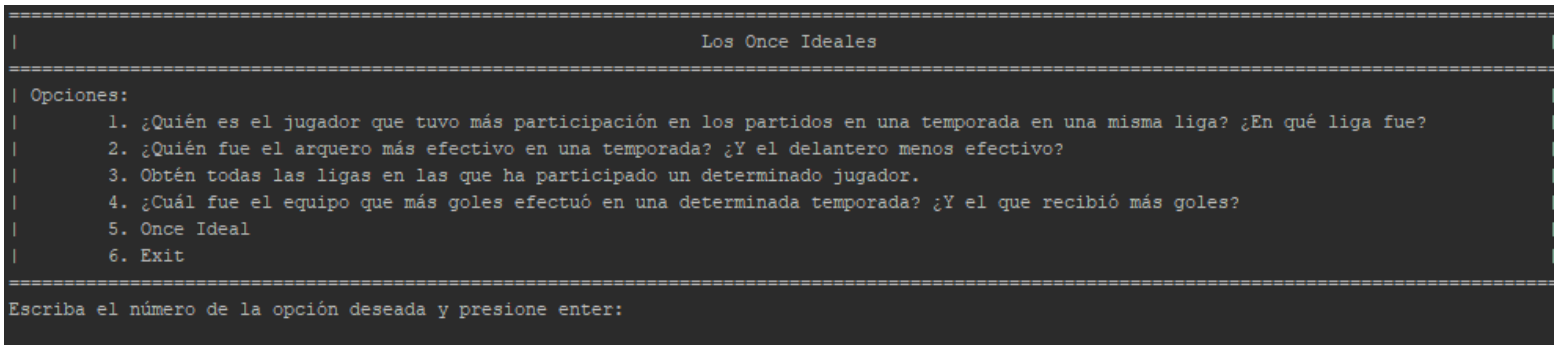
Por no tener la parte 2a realizada, no se cuenta con un golero.

Se tomó una formacion 4-3-3.

Conclusiones

Podemos concluir que en base a por como se crearon las clases y la manera de almacenar los datos este programa va a encontrar las respuestas a todas las preguntas de una manera rápida y concreta, manteniendo buenos criterios y utilizando los TDA provistos.

Guía del usuario



Se le presenta al usuario un menú principal.

Aquí usted debe optar por una de las opciones disponibles, indicadas por un número particular.

Como se muestra debe presionar el numero que desea y después presionar enter.

Aquí dependiendo de cada opción se le puede solicitar información adicional.

Tambien al culminar cada pregunta se podrá volver a seleccionar.

Para salir presione 6.