Resumen

Resumen de los objetivos de la práctica

Objetivos

- Conocer el método de resolución algorítmica Divide y Vencerás
- Realizar un análisis teórico de la complejidad de la solución aportada.
- Desarrollar una solución basada en la técnica de Divide y Vencerás
- Comprobar de manera empírica los resultados obtenidos en el problema.
- Diseñar mejoras sobre la primera solución aportada.

1. Introducción

Queremos zanjar una pregunta que se hacen los analistas de EEUU desde tiempos inmemoriales, quienes son los mejores jugadores de la NBA de todos los tiempos. Para ello contamos con un conjunto de datos de las estadísticas de todos los jugadores desde el año 1950 al 2017. De cada jugador contamos con las siguientes estadísticas de cada uno en sus años en activo:

- `#`: Número identificativo de un jugador durante un año concreto (No nos interesa)
- `SeasonStart`: Año de la temporada.
- `PlayerName`: Nombre del jugador.
- `PlayerSalary`: Salario del jugador un año específico.
- `Pos`: Posición del jugador.
- `Age`: Edad del jugador.
- `TM`: Equipo.
- `FG%`: Porcentaje de aciertos en tiros de campo.
- `PTS`: Puntos anotados en la temporada.

NbaStats.csv

```
#;SeasonStart;PlayerName;PlayerSalary;Pos;Age;Tm;FG%;PTS
8035;1986;A.C. Green;;PF;22;LAL;53,9;521
8420;1987;A.C. Green;;PF;23;LAL;53,8;852
8807;1988;A.C. Green;;PF;24;LAL;50,3;937
9242;1989;A.C. Green;;PF;25;LAL;52,9;1088
9688;1990;A.C. Green;$1,750,000.00 ;PF;26;LAL;47,8;1061
10166;1991;A.C. Green;$1,750,000.00 ;PF;27;LAL;47,6;750
10617;1992;A.C. Green;$1,750,000.00 ;PF;28;LAL;47,6;1116
11060;1993;A.C. Green;$1,885,000.00 ;PF;29;LAL;53,7;1051
11529;1994;A.C. Green;$6,472,600.00 ;PF;30;PHO;50,2;1204
11999;1995;A.C. Green;$6,473,000.00 ;SF;31;PHO;50,4;916
12477;1996;A.C. Green;$4,851,000.00 ;SF;32;PHO;48,4;612
13034;1997;A.C. Green;;PF;33;TOT;48,3;597
13035;1997;A.C. Green;;PF;33;PHO;47,7;153
13036;1997;A.C. Green;;PF;33;DAL;48,6;444
13600;1998;A.C. Green;$5,125,088.00 ;PF;34;DAL;45,3;600
14139;1999;A.C. Green;$1,700,000.00 ;PF;35;DAL;42,2;246
14648;2000;A.C. Green;$2,250,000.00 ;PF;36;LAL;44,7;413
15148;2001;A.C. Green;;PF;37;MIA;44,4;367
14526;2000;A.J. Bramlett;;C;23;CLE;19;8
10130;1991;A.J. English;$325,000.00 ;SG;23;WSB;43,9;616
10583;1992;A.J. English;$406,000.00 ;SG;24;WSB;43,3;886
15151;2001;A.J. Guyton;$465,850.00 ;PG;22;CHI;40,6;198
15684;2002;A.J. Guyton;$18,748.00 ;PG;23;CHI;36,1;244
16172;2003;A.J. Guyton;;PG;24;GSW;0;0
```

Figure 1. Stats de los jugadores de la NBA

NOTE En la imagen podéis observar que los datos están separados por ";". Por otro lado, la primera

Nuestro objetivo es hallar los 10 mejores jugadores de todos los tiempos aplicando la técnica de Divide y Vencerás para encontrar la solución.

2. A tener en cuenta

- Los datos están disponibles en el repositorio GitHub de esta práctica `NbaStats.csv`.
- Al cargar los datos generamos un registro para cada jugador, no múltiples, por lo que tendremos que comprobar si el jugador ya está introducido en la estructura de datos que estemos utilizando o no.
- Para la carga de los datos utilizaremos una clase POJO (Plain Old JAVA Object) denominada Player, parecida a la siguiente.

Example 1. Player.java source code

```
public class Player{

    private String playerName;
    private ArrayList<String> teams;
    private ArrayList<String> positions;
    private int score;

    public Player(String playerName, String team, String position, int
score){
        this.playerName = playerName;
        this.teams = new ArrayList<String>();
        this.teams.add(team);
        this.positions = new ArrayList<String>();
        this.positions.add(position);
        this.score = score;
    }

    public String getPlayerName(){
        return this.playerName;
    }
    public void setPlayerName(String playerName){
        this.playerName = playerName;
    }

    public ArrayList<String> getTeams(){
        return this.teams;
    }
    public void setTeams(ArrayList<String> teams){
        this.teams = teams;
    }

    public ArrayList<String> getPositions(){
        return this.positions;
    }
    public void setPositions(ArrayList<String> positions){
        this.positions = positions;
    }

    public int getScore(){
        return this.score;
    }
    public void setScore(int score){
        this.score = score;
    }
}
```

- El atributo `score` coincide con los puntos metidos durante un año multiplicado por FG%/100, ya que no queremos jugadores que metan muchos puntos pero lo fallen casi todo.
- Si el jugador está ya añadido en la estructura de datos que deseamos utilizar para resolver el problema añadiremos el equipo en la variable `teams`, añadiremos la posición en la variable `positions` y haremos la media del `score` que ya haya y el `score` correspondiente en el nuevo año.

3. Trabajo a desarrollar

Debéis proponer e implementar una solución recursiva basada en el esquema general de DyV que devuelva los 10 mejores jugadores de todos los tiempos si tan solo tenemos en cuenta su score. Realizaremos un análisis de la complejidad del algoritmo utilizando el método maestro. Por otro lado, realizaremos una comprobación de los tiempos de ejecución del o de los algoritmos que hayas desarrollado.

Valoraremos las mejoras que desarrolléis a posteriori siempre que las justifiquéis de manera empírica en la puntuación de la práctica, aunque no sea estrictamente necesaria para aprobar la práctica.

4. Entregas

Se ha de entregar, en fecha, un repositorio público de GitHub con toda la documentación y código fuente requerido en la práctica:

- En el repositorio crear una carpeta llamada practica_1, donde creeis dos subcarpetas una para la documentación, `docs` y otra para el código fuente `sources`.
- Memoria o presentación que explique los distintos algoritmos que habéis utilizado para resolver el problema teniendo en cuenta el análisis de eficiencia así como la identificación de cada una de las partes del esquema general de divide y vencerás.
- Código fuente de la aplicación, desarrollada en JAVA, que resuelva el problema planteado. Tendréis que medir el tiempo de ejecución de vuestra solución por lo que podréis incluir las órdenes necesarias para ello en el código fuente.
- Juegos de prueba que consideréis oportunos para asegurarnos que la aplicación funciona como es debido.

DOCUMENTACION DEL PROBLEMA A RESOLVER Y METODOS EMPLEADOS

Para este problema hemos planteado la lista de jugadores con sus respectivos parámetros de tal forma que los mejores jugadores serán aquellos que mejor media tengan durante sus años en activo, es decir la media de sus puntuaciones anuales partido el número de años. Para ello se ha optado por la implantación de una ArrayList que contenga los datos del fichero NbaStats.csv a la que se le añade también la media calculada de cada jugador para así poder completar el pretratamiento de los datos, de tal forma que antes de aplicar el algoritmo y tengamos en una misma estructura todo lo necesario para la correcta obtención de los mejores jugadores de la NBA.

Para ordenar dichos datos y obtener el resultado deseado implementaremos un Quicksort a dicha ArrayList por el cual mediante un pivote y un índice izquierdo (recorre de izq a der) y otro índice derecho (recorre de der a izq) recorreremos las posiciones de la lista dividiéndola en sublistas hasta hallar las mejores medias cerciorándonos que las búsquedas no se cruzan en ningún momento de esta forma tendremos que ir comprobando subArrayList izquierdo y derecho respecto al pivote hasta recorrer toda la ArrayList pasada como parámetro.

Debido a que solo queremos los diez mejores jugadores, como mejora para el algoritmo se planteó una comprobación extra. Si la sublista derecha en la que se hayan los valores mayores que el pivote (valor de la media del jugador) alcanzaba este valor, no era necesaria comprobar la izquierda ya que allí solo habría valores menores es decir valores de media menores a la del pivote y en definitiva peores jugadores (siguiendo la media como criterio). Por lo que se mejoraría la eficiencia de obtención del resultado del problema por parte del Quicksort. A parte al estar recorriendo sublistas es más ágil que intentar comprobar uno a uno los valores como si se tratase de la obtención de un máximo ya que es bastante a recorrer permitiéndonos así poder parar la búsqueda si ya tenemos a los 10 mejores sin necesidad de recorrer la lista hasta el final.

Algoritmos Empleados:

Quicksort: Para la ordenacion y division del ArrayList de jugadores y así obtener los diez mejores con la mejora de comprobación de tamaño del subArrayList derecho con respecto al pivote para comprobar si ya tenemos a los diez mejores y detener la ejecución ahí. Siendo el orden de complejidad de este algoritmo de O **$(n \cdot \log n)$.**

-Siendo el arraylist el problema principal a tratar.

-Los subarrays derecho e izquierdo los subproblemas.

-Los 10 jugadores nuestro umbral optimo.

-las llamadas de Quicksort para descomponer en subproblemas el arraylist inicial.

-Debido a la mejor implementado no habría necesidad de combinar las soluciones ya que se encuentran directamente en el subarraryList derecho.

Captura de pantalla de la ejecución del algoritmo con los tiempos de ejecución de varias iteraciones del mismo y con la lista de los 10 mejores jugadores obtenidos se presenta a continuación: