

Estructuras de Datos

# **Práctica 1**

# **Manual Tecnico**

---

Fernando Jose Rodriguez Ramirez - 202030542



## Introducción

El programa de gestión de apuestas del hipódromo es un programa altamente eficiente que permite el procesamiento e inicio de carreras. El funcionamiento de los algoritmos empleados se explicará en las siguientes secciones.

## Primer Servicio: Ingreso de Apuestas

Para el ingreso de las apuestas el input es un archivo de texto CSV y el output es un arreglo de Strings que fue procesado en dos pasos utilizando las librerías estándar de Java para Strings.

```
public String[] ingresoApuestas(Path directorioArchivo) throws IOException { //Dos pasos O(1)
    String contenidoArchivo = Files.readString(directorioArchivo);
    String[] apuestas = contenidoArchivo.split("\r|\r\n|\n|(\s)*,(\s)*");
    return apuestas;
}
```

El archivo de texto es convertido en un String que posteriormente es dividido cuando se encuentre con saltos de línea y con valores separados por coma haciendo que cada dato se convierta en un String de dicho arreglo.

Se evitó un mayor procesamiento de los datos para evitar un mayor tiempo de ejecución.

Al contar con únicamente dos pasos sin importar el tamaño del archivo se considera que la complejidad del algoritmo es de:  **$O(1)$**

## Segundo Servicio: Verificación de Apuestas

El segundo servicio crítico es el servicio de verificación de las apuestas ingresadas, este servicio se encarga de almacenar las apuestas que fueron correctamente ingresadas y las que no.

El algoritmo tiene como input un arreglo de Strings que contienen todos los datos de las apuestas y como output una lista enlazada con las apuestas que se aprobaron y un archivo de texto con todas las apuestas rechazadas.

```
public ListaApuestas[] verificacionApuestas(String[] datosApuestas) throws IOException { //Un ciclo, 0(n)
    FileWriter myWriter = new FileWriter(System.getProperty("user.home")+"/apuestas_rechazadas.txt");

    ListaApuestas listaApuestasAprobadas = new ListaApuestas();
    ListaApuestas listaApuestasRechazadas = new ListaApuestas();
    String erroresLista = "";
    String apostador = null;
    float monto = 0;
    int[] posicionesCaballos = new int[10];
    int i=0; //Representa el tipo de dato de la apuesta: 0=nombre, 1=monto, 2-11=posiciones_carreza


    for (String datoApuesta : datosApuestas) {
        int pasos = 0;
        Instant starts = Instant.now(); pasos++;
        if (i==0) {
            apostador = datoApuesta; pasos++;
        }
        if (i==1) {
            monto = Float.parseFloat(datoApuesta).pasos++;
        }
        if (2<=i && i<=11) {
            int posicionCaballo = i-1; //Representa la posicion que el apostador cree que terminara un caballo
            int numeroDeCaballo = Integer.parseInt(datoApuesta); //Representa el numero de caballo por el que se apuesta
            pasos+=2;
            if(posicionesCaballos[numeroDeCaballo-1] == 0){
                posicionesCaballos[numeroDeCaballo-1] = posicionCaballo.pasos++;
            } else {
                erroresLista += "Se ingreso mas de una vez el caballo "+String.valueOf(numeroDeCaballo)+"\n".pasos++;
            }
        }
        if (i==11) {
            if (erroresLista.equals("")) {
                listaApuestasAprobadas.agregar(new Apuesta(apostador, monto, posicionesCaballos, erroresLista)).pasos++;
            } else {
                myWriter.write(new Apuesta(apostador, monto, posicionesCaballos, erroresLista).toString());
                myWriter.write("-----");
                pasos+=2;
            }
            i=0;
            erroresLista = "";
            posicionesCaballos = new int[10];
            pasos+=3;
        } else {
            i++;pasos++;
        }
        Instant ends = Instant.now();
        sumaTiempoVerificacionApuestas += Duration.between(starts, ends).toNanos();
        sumaPasosVerificacionApuestas += pasos;
        cantidadApuestas++;
        if (pasos>mayorCantidadPasos) {mayorCantidadPasos = pasos;}
        if (pasos<menorCantidadPasos || menorCantidadPasos==1) {menorCantidadPasos = pasos;}
    }
    myWriter.close();

    double promedioTiempo = sumaTiempoVerificacionApuestas/cantidadApuestas;
    float promedioPasos = sumaPasosVerificacionApuestas/cantidadApuestas;

    ManejadorReportes crearReporteVerificacionApuesta(
        promedioTiempo,
        promedioPasos,
        mayorCantidadPasos, menorCantidadPasos
    );

    return new ListaApuestas[]{listaApuestasAprobadas, listaApuestasRechazadas};
}
```

El algoritmo recorre el arreglo de Strings con los datos de las apuestas, debido a que los datos son de diferente tipo siendo estos los nombres de los apostadores, el monto apostado y las posiciones de los caballos la forma en que se mantiene el control sobre los tipos de datos es por medio de un contador ya que se sabe que todas las apuestas van a tener una estructura donde, la primera posición será el nombre del apostador, la segunda posición el monto apostado y las posiciones de la tercera a la 12 serán las posiciones de los caballos. el contador aumenta en uno



cada que se cambia el dato, reiniciando cuando el contador llega a la posición 12. Gracias a ese sistema el algoritmo mantiene una máxima eficiencia.

Cuando se asignan las posiciones de los caballos se hace de modo que el número de caballo representa su posición en el arreglo, haciendo que por ejemplo si el caballo 7 está en primer lugar, el arreglo en la posición 7 tendrá un 1 (se toma en cuenta que los arreglos empiezan en 0), por ello para evitar duplicados se verifica que el espacio para el caballo aun no esta ocupado, si lo esta significa que un caballo esta duplicado.

Se usó este método para evitar utilizar un ciclo que tuviese que recorrer el arreglo cada vez que se introducía un dato en este, aumentando la eficiencia.

Para el manejo de errores el algoritmo encontrará todos los errores en una apuesta y no rechazara la apuesta de inmediato, hasta alcanzar la posición 12.

Debido a que el algoritmo únicamente usa un ciclo para procesar los datos la complejidad del algoritmo es:  **$O(n)$**

## Servicio Tres: Cálculo de Resultados

El servicio que calcula los resultados se encarga de verificar todas las apuestas para puntuar a cada uno de los apostantes. Para eso utiliza un método recursivo que recorre cada uno de los nodos de la lista enlazada donde cada nodo contiene una apuesta.

El input del algoritmo es un arreglo con los resultados de la carrera y la lista de apuestas realizadas.

```
public void verificarResultados(int[] resultadosCarrera){
    verificarResultadoNodo(head, resultadosCarrera);
    ManejadorReportes.crearReporteCalculoResultados(
        sumaTiempoVerificacionResultados/cantidadApuestas,
        sumaPasosVerificacionResultados/cantidadApuestas,
        mayorCantidadPasosResultados, menorCantidadPasosResultados
    );
}

private void verificarResultadoNodo(Nodo nodoApuesta, int[] resultadosCarrera){
    int pasos = 0;
    Instant starts = Instant.now();
    Apuesta apuesta = nodoApuesta.informacion;
    int[] ordenCaballos = apuesta.getOrdenCaballos();
    for (int i = 0; i < 10; i++) {
        if (ordenCaballos[resultadosCarrera[i]-1]==(i+1)) {
            apuesta.setPunteoApuesta(apuesta.getPunteoApuesta()+10-i);
        }
        pasos++;
    }
    Instant ends = Instant.now();
    sumaTiempoVerificacionResultados += Duration.between(starts, ends).toNanos();
    sumaPasosVerificacionResultados += pasos;
    cantidadApuestas++;
    if (pasos>mayorCantidadPasosResultados) {mayorCantidadPasosResultados = pasos;}
    if (pasos<menorCantidadPasosResultados || menorCantidadPasosResultados==-1) {menorCantidadPasosResultados = pasos;}

    if (nodoApuesta.getSiguiente()!=null) {
        verificarResultadoNodo(nodoApuesta.getSiguiente(), resultadosCarrera);
    }
}
```

Dentro del método recursivo hay un ciclo que siempre se ejecuta 10 veces, este ciclo se encarga de conocer si el apostante acierta en sus predicciones, para esto se toma en cuenta que por ejemplo el índice 6 del arreglo contiene la posición del caballo 7, mientras que el arreglo de los resultados en caso que el caballo 7 este en el índice 0 significa que este gano, así que por medio de una conversión entre índice y posición se puede saber en que posición quedó cada caballo.

Se usó este método para evitar el uso de más ciclos en el servicio anterior.

Ya que el método recursivo se ejecuta  $n$  veces y debido a que el ciclo siempre se ejecuta solamente 10 veces la complejidad del algoritmo es:  **$O(n)$**

## Servicio Cuatro: Ordenamiento de Resultados

El servicio de ordenamiento de resultados debe de ser capaz de ordenar las apuestas en cuántos puntos obtuvo cada una y también en orden alfabético con los nombres de los apostantes. Para esto se eligió el método burbuja, ya que es un método de ordenamiento relativamente eficiente al mismo tiempo que es fácil de programar.

```
public void ordenarPorPunteo(){ //Dos ciclos, O(n^2)
    Instant starts = Instant.now();
    int sizeLista = size();
    if (sizeLista > 1) {
        for (int i = 0; i < sizeLista; i++) {
            Nodo nodoActual = head;
            Nodo next = head.getSiguiente();
            sumaPasosOrdenamientoPunteo+=2;
            for (int j = 0; j < sizeLista - 1; j++) {
                if (nodoActual.informacion.getPunteoApuesta() > next.informacion.getPunteoApuesta()) {
                    swapNodos(nodoActual, next, false);
                    sumaPasosOrdenamientoPunteo+=4;
                }
                nodoActual = next;
                next = next.getSiguiente();
                sumaPasosOrdenamientoPunteo+=2;
            }
        }
    }
    Instant ends = Instant.now();
    sumaTiempoOrdenamientoPunteo = Duration.between(starts, ends).toNanos();
    ManejadorReportes.crearReporteOrdenamientoPunteo(sumaTiempoOrdenamientoPunteo/cantidadApuestas,
                                                    sumaPasosOrdenamientoPunteo/cantidadApuestas
                                                    );
}

public void ordenarAlfabeticamente(){
    Instant starts = Instant.now();
    int sizeLista = size();
    if (sizeLista > 1) {
        for (int i = 0; i < sizeLista; i++) {
            Nodo nodoActual = head;
            Nodo next = head.getSiguiente();
            sumaPasosOrdenamientoAlfabetico+=2;
            for (int j = 0; j < sizeLista - 1; j++) {
                if (nodoActual.informacion.getApostador().compareTo(next.informacion.getApostador()) > 0) {
                    swapNodos(nodoActual, next, true);
                    sumaPasosOrdenamientoAlfabetico+=4;
                }
                nodoActual = next;
                next = next.getSiguiente();
                sumaPasosOrdenamientoAlfabetico+=2;
            }
        }
    }
    Instant ends = Instant.now();
    sumaTiempoOrdenamientoAlfabetico = Duration.between(starts, ends).toNanos();
    ManejadorReportes.crearReporteOrdenamientoAlfabetico(
                                                    sumaTiempoOrdenamientoAlfabetico/cantidadApuestas,
                                                    sumaPasosOrdenamientoAlfabetico/cantidadApuestas
                                                    );
}
```

El ordenamiento burbuja utiliza un método de la lista que se encarga de obtener el tamaño de esta, dicho método tiene un método recursivo que se ejecuta n veces, ya que su objetivo es determinar cuántos elementos tiene el arreglo.

```

public int cantidadSiguients(){
    if (getSiguiente()==null) {
        return 1;
    } else {
        return getSiguiente().cantidadSiguients()+1;
    }
}

```

El método de ordenamiento burbuja utiliza el método compareTo de los Strings para saber cuales nombres van en qué posición alfabéticamente.

El ordenamiento burbuja necesita hacer cambios dentro del orden de los nodos del arreglo por lo que se utiliza el método swapNodos.

```

public void swapNodos(Nodo primerNodo, Nodo segundoNodo, boolean alfabetico){
    if (primerNodo.getAnterior()!=null) {
        primerNodo.getAnterior().setSiguiente(segundoNodo);
        if (alfabetico) {sumaPasosOrdenamientoAlfabetico++;}
        else {sumaPasosOrdenamientoPunteo++;}
    }
    if (segundoNodo.getSiguiente()!=null) {
        segundoNodo.getSiguiente().setAnterior(primerNodo);
        if (alfabetico) {sumaPasosOrdenamientoAlfabetico++;}
        else {sumaPasosOrdenamientoPunteo++;}
    }

    segundoNodo.setAnterior(primerNodo.getAnterior());
    primerNodo.setSiguiente(segundoNodo.getSiguiente());
    segundoNodo.setSiguiente(primerNodo);
    primerNodo.setAnterior(segundoNodo);

    if (primerNodo == head) {
        head = segundoNodo;
        if (alfabetico) {sumaPasosOrdenamientoAlfabetico++;}
        else {sumaPasosOrdenamientoPunteo++;}
    }
    if (segundoNodo==tail) {
        tail = primerNodo;
        if (alfabetico) {sumaPasosOrdenamientoAlfabetico++;}
        else {sumaPasosOrdenamientoPunteo++;}
    }
}

```

El método cambia las referencias de los nodos cambiando la posición de dos nodos en la lista, el tiempo de este método es constante.

Tomando en cuenta que el método para obtener el tamaño de la lista tiene tiempo lineal, el método que cambia los nodos tiene tiempo constante y el método burbuja utiliza dos ciclos podemos deducir que el tiempo del método es  $n^2+n+1$  que puede resumirse en:  **$O(n^2)$**