Universidad de San Carlos de Guatemala

Centro Universitario de Occidente

División Ciencias de la Ingeniería

Estructura de datos

Ing. Oliver Ernesto Sierra Pac

Aux. Fátima Tezo

Primera práctica

Nombre:

Pedro Ricardo Gordillo González

Registro Académico:

202031683

Quetzaltenango, marzo 2022

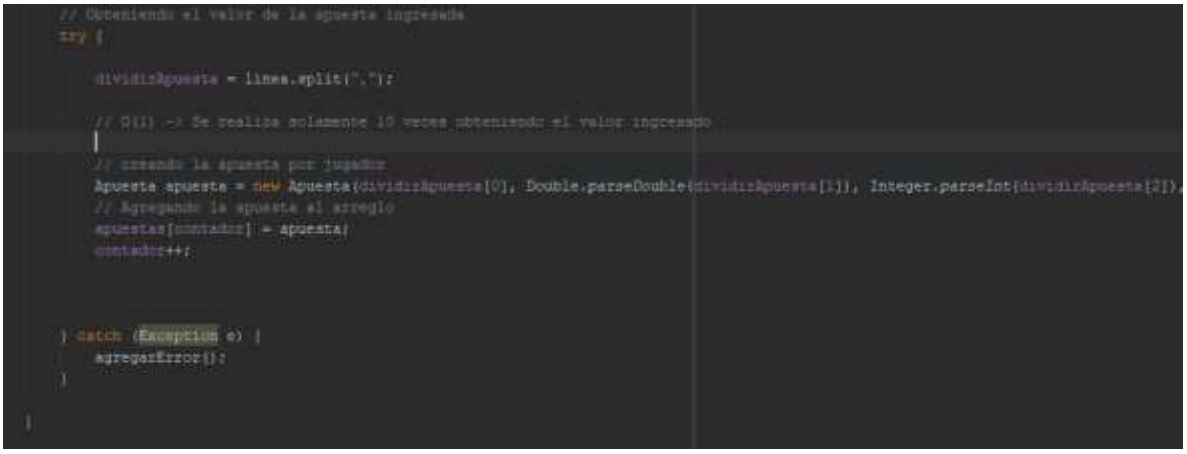
Manual Técnico

Ingreso de las apuestas antes del inicio de la carrera

El ingreso de las apuestas tiene una complejidad de O(1), esto ya que se obtiene un numero contable de veces (10 veces) los valores que ingresan de las apuestas en el archivo cargado para obtener los datos del jugador.

Cálculos

Split	Lectura de la separación de los datos	O(10)
Creación de array	Asignación de los datos de la apuesta	O(10)
	Complejidad	O(1)




```
try{

// Carga de las apuestas
 ManejoArchivo cargar = new ManejoArchivo();
 cargar.leerLinea();
 apuesta = cargar.getApuestas();
 contador = cargar.getContador();
 errores = cargar.getErrores();

 }catch(Exception e){
     JOptionPane.showMessageDialog(null,"No se selecciono ningun archivo");
 }
```

Cerrar apuestas

Se cierran las apuestas que se tienen y se continua con la verificación del ingreso de los resultados de la carrera.

Se realiza de esta manera para llevar un control y trabajar únicamente con una referencia a memoria, es decir, con un único arreglo para ahorrar recursos.

ver2 = new VerificarApuesta()	Se crea la referencia a la clase que realizara la verificación de apuestas.	O(1)
ver2.verificarApuestas	Se realiza la verificación de las apuestas en la clase, como se explica posteriormente.	O(1)
ver2.correctas	Se retorna y se asigna el arreglo con las apuestas correctas.	O(1)
ver2.actualizar	Se actualiza el reporte de las apuestas.	O(1)
	Complejidad	O(1)

```
    //Se verifican las apuestas cargas
    VerificarApuestas ver2 = new VerificarApuestas();
    Apuesta[] corregidas =ver2.verficadorApuestas(apuesta, contador,errores,t);
    Apuesta[] correctas =ver2.correctas(corregidas);
    Reportes reportel = ver2.actualizarReporte(reporte);


    //Se les da seguimiento a las apuestas correctas
    IngresarPosiciones posiciones = new IngresarPosiciones(correctas,reportel);
    this.setVisible(false);
    posiciones.setVisible(true);
```

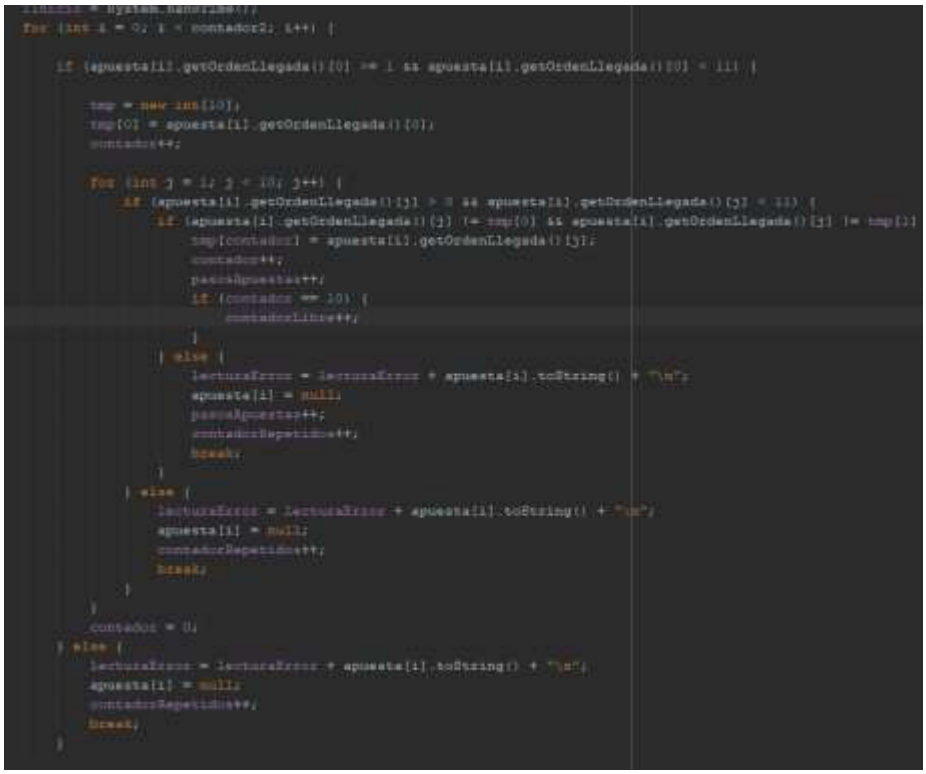
Verificación de apuestas

Al realizar la verificación de apuestas se utiliza un " for " que depende del tamaño de las apuestas ingresadas, dentro hay otro " for " que no es considerado con complejidad n ya que solamente depende de un iterador de tamaño 10, al ingresar al for de iterador constante se realiza una verificación para comprobar que el numero de caballo ingresado en la apuesta no sea menor a 1 ni mayor a 10, si se cumple con esto, se detecta como error y se envía al historial de errores.

Al encontrar una repitencia de caballos en la misma apuesta se marca como nulo el array y se agrega al historial de errores.

Cálculos

For (iterador variable)	Accede a cada apuesta ingresada.	O(n)
If (primer caballo)	Accede a la primera posición del caballo dentro de la apuesta.	O(1)
For (iterador constante 10)	Accede a cada posición de la apuesta ingresada por el jugador.	O(10)
If (posición caballo)	Verifica que la posición actual del caballo no sea menor a 1 ni mayor a 10.	O(10)
If (posición caballo)	Accede las 10 posiciones de la apuesta para verificar que no exista ninguna repetida.	O(10)
Else	Si el numero de la apuesta del caballo es mayor a 10 o menor a 0 se detecta como error	O(10)
	Complejidad	O(n)



Apuestas correctas

Este método se encarga de verificar si las apuestas son nulas, si son nulas significa que son descartadas, se crea un array de apuestas para ingresar solamente las correctas y se deja de dar seguimiento a las apuestas con algún tipo de error. Se realizo de esta manera para aprovechar que solamente se deben de compara posición por posición y así mismo poder aprovechar que si esta correcta la apuesta se va restando solamente las posiciones pasadas.

lib = new Apuesta	Se crea el objeto que contendrá las nuevas apuestas.	O(1)
For (iterador variable)	Se recorre un iterador para verificar e ingresar las apuestas distintas de null a las cuales se les dará seguimiento.	O(n)
If (verificación)	Se verifica que la apuesta en la posición del iterador sea distinta de null.	O(1)
	Complejidad	O(n)

```
public Apuesta[] correctas(Apuesta[] corregir) {

    int nu = 0;
    Apuesta[] lib = new Apuesta[contadorLibre];

    for (int i = 0; i < corregir.length; i++) {
        if (corregir[i] != null) {
            lib[nu] = corregir[i];
            nu++;
        }
    }

    return lib;
}
```

Cálculo de tiempo y promedio

Se realizo de esta manera para únicamente trabajar con una referencia a memoria la cual contendrá los valores finales al culminar el programa su ejecución.

If (comparacion mayor a 0)	Si el tamaño del array es mayor a cero se calcula lo solicitado.	O(1)
else (comparación menor a 0)	Si el tamaño es cero, se retorna solamente el tiempo de ejecución del método utilizado.	O(1)

```
public long getTtotalApuestas() {
    if (tamañoApuestas > 0) {
        tiempoApuestas = (TFinal - TInicio) / tamañoApuestas;
    } else {
        tiempoApuestas = (TFinal - TInicio);
    }

    return tiempoApuestas;
}

public int getPromedioPasos() {

    if (tamañoApuestas > 0) {
        promedioPasos = pasosApuestas / tamañoApuestas;
    } else {
        promedioPasos = pasosApuestas;
    }
    return promedioPasos;
}
```

Actualizar reporte de apuestas

Según los datos obtenidos de los cálculos se realiza la actualización a los reportes de las apuestas. Esto se realiza de esta manera para aprovechar que se está trabajando únicamente con una referencia a memoria.

reporte.setTiempoApuestas()	Se ingresa el dato obtenido al reporte.	O(1)
reporte.setPasosApuestas()	Se ingresa el dato obtenido al reporte.	O(1)
reporte.setMaxApuestas()	Se ingresa el dato obtenido al reporte.	O(1)

reporte.setMinApuestas()	Se ingresa el dato obtenido al reporte.	O(1)
	Complejidad	O(1)

```
public Reportes actualizarReporte(Reportes reporte){

    reporte.setTiempoApuestas(getTtotalApuestas());
    reporte.setPasosApuestas(getPromedioPasos());
    reporte.setMaxpasosApuestas(0);
    reporte.setMinpasosApuestas(0);

    return reporte;
}
```

Verificar resultados

Al ingresar los resultados de la carrera, se realiza una verificación donde no pueden existir dos posiciones iguales, para ello se realiza lo siguiente. Es solamente un intermedio entre la verificación y la interfaz, se detalla a continuación.

If(verificar repitencia == false)	Se realiza una comparación si hay repetidos dentro de las posiciones ingresadas de la carrera.	O((1)
ganancia = new GenerarGanacia()	Si no hay repitencia en las posiciones del resultado de la carrera se realiza el calculo del puntaje del jugador.	O((1)
reporte2 = ganancia.actualizacionRep()	Se realiza la actualización de los reportes.	O((1)

```
    if (!pos.verificarRepitencia(posiciones)) {
        GenerarGanancia ganancia = new GenerarGanancia();
        ganancia.getResultados(apuestas, posiciones);
        Reportes reporte2 = ganancia.actualizarReportes(reportes);
```

```
} else {
    JOptionPane.showMessageDialog(null, "Las posiciones no pueden ser repetidas y estar en el rango de 1 a 10");
}
```

Verificación de repitencia en los resultados de la carrera

Al ingresar los resultados de la carrera se realiza la verificación para que no existan posiciones repetidas. Se realizo así aprovechando que al existir solo una repitencia ya se considerara como error.

For (interador i constante)	Se recorre cada posición del arreglo de las “posiciones”.	O(1)
If (verificación)	Se verifica que la posición actual no sea mayor a 10 o menor a 1.	O(1)
For (interador j constante)	Se recorre cada posición + 1 del arreglo de las “posiciones”.	O(1)
If (verificación)	Se verifica que la posición actual no sea igual a la posición siguiente.	O(1)
Else	Si los resultados son iguales se retorna false y se marca error.	O(1)
	Complejidad	O(1)

```
public class verificarPosiciones {

    public boolean verificarRepitencia(int[] resultados) {

        for (int i = 0; i < 10; i++) {

            if (resultados[i] > 0 && resultados[i] < 11) {

                for (int j = i + 1; j < 10; j++) {

                    if (resultados[j] > 0 && resultados[j] < 11) {

                        if (resultados[i] == resultados[j]) {
                            return true;
                        }
                    } else {
                        return true;
                    }
                }
            }else{
                return true;
            }

        }

        return false;
    }
}
```

Calcular puntos

Si las posiciones del resultado de la carrera son distintos se realiza el cálculo del puntaje de cada jugador según las posiciones que ingreso. Se realiza de esta manera previendo que la complejidad debe de ser como máximo O(n), con ello se aprovecha que la cantidad de resultados son constantes, es decir, no varían en tamaño.

for (iterador i variable)	Recorre cada posición de la apuesta realizada.	O(n)
for (iterador j constante)	Recorre cada resultado y cada posición de la apuesta para comparar si son iguales.	O(10)
if (comparación)	Compara si la posición del resultado es igual a la de la apuesta. Si es correcta realiza la suma de la ganancia, 10 puntos y resta la posición evaluada.	O(10)
else	Si no es igual la posicion del resultado y la de la apuesta no se suman puntos.	O(10)
	Complejidad	O(n)

```
public Apuesta[] getResultados(Apuesta[] apuestas, int[] resultados) {
    tInicial=System.nanoTime();
    this.tamaño = apuestas.length;

    for (int i = 0; i < apuestas.length; i++) {

        int ganancia = 0;

        for (int j = 0; j < 10; j++) {
            if (apuestas[i].getOrdenLlegada()[j] == resultados[j]) {
                ganancia = ganancia + mayorGanancia - j;
                pasos++;
            } else {
                ganancia = ganancia + 0;
                pasos++;
            }
        }
        apuestas[i].setGanancia(ganancia);
    }
    tFinal = System.nanoTime();

    return apuestas;
}
```

Calcular promedios

Según el tiempo que transcurre desde el inicio hasta el final se calcula un promedio con la ayuda del tamaño del arreglo, con esto se lleva a cabo la comparación de las posiciones tanto del jugador como del resultado. Como se mencionaba anteriormente, se realiza así para trabajar únicamente con un solo objeto.

If (comparación)	Compara si la longitud de las apuestas es mayor a cero, si sí se realiza el calculo tomando en cuenta el tamaño del array.	O(1)
Else	Si la longitud del arreglo es cero se retorna solamente el tiempo y la cantidad de pasos.	O(1)
	Complejidad	O(1)

```
public long getTiempoGanancias(){

   if(tamaño>0){
       tiempoGanancias = (tFinal-tInicial)/tamaño;
   }else{
       tiempoGanancias = (tFinal-tInicial);
   }

 return tiempoGanancias;
}

public int getPasosGanancias(){

    if(tamaño>0){
        pasosProm = pasos/tamaño;
   }else{
        pasosProm = pasos;
   }
    return pasosProm;
}
```

Actualización de reportes

Según los datos obtenidos de los cálculos se realiza la actualización a los reportes de los resultados.

reporte.setTiempoResultados()	Se ingresa el dato obtenido al reporte.	O(1)
reporte.setPasosResultados ()	Se ingresa el dato obtenido al reporte.	O(1)
reporte.setMaxResultados()	Se ingresa el dato obtenido al reporte.	O(1)
reporte.setMinResultados ()	Se ingresa el dato obtenido al reporte.	O(1)
	Complejidad	O(1)

```
public Reportes actualizarReportes(Reportes reporte){

    reporte.setTiempoResultados(getTiempoGanancias());
    reporte.setPasosResultados(getPasosGanancias());
    reporte.setMaxpasosResultados(0);
    reporte.setMinpasosResultados(0);

    return reporte;
}
```

Ordenar por Puntaje obtenido

Al seleccionar la opción del puntaje obtenido se realiza el ordenamiento por medio del método burbuja.

Argumentación

Utilice el método de burbuja para realizar el ordenamiento ya que según documentación el método de ordenación por inserción como “desventaja es que no funciona tan bien como otros algoritmos mejores de ordenamiento. Con n al cuadrado pasos requeridos para cada n elemento a ser ordenado, este algoritmo no funciona bien con una lista grande. Por lo tanto, este sólo es útil cuando se ordena una lista de pocos elementos”, y, siendo el caso que el programa se trata de apuestas es sabido que en ese tipo de juegos de azar son grandes cantidades creo que es más conveniente utilizar el método burbuja para el ordenamiento.

https://techlandia.com/ordenar-lista-enlazada-java-como_47341/

```java
public Apuesta[] OrdenDescendente(Apuesta[] apuesta) {
    Apuesta auxiliar;
    this.tamaño = apuesta.length;
    tInicial = System.nanoTime();
    for (int i = 0; i < (apuesta.length - 1); i++) {

        for (int j = 0; j < (apuesta.length - 1); j++) {
            if (apuesta[j].getGanancia() < apuesta[j + 1].getGanancia()) { // si numero actural m
                auxiliar = apuesta[j];
                apuesta[j] = apuesta[j + 1];
                apuesta[j + 1] = auxiliar;
                pasos++;
            }
        }
    }
    tFinal = System.nanoTime();
    return apuesta;
}
```

for(i variable)	Accede a cada puntaje de apuesta realizada	O(n)
for(j variable)	Accede a cada puntaje -1 de apuesta realizada	O(n)
If	sí número actual mayor numero siguiente se realiza el cambio	O(1)
Else (implícito)	No se realiza el cambio	O(1)
	Complejidad	O(n^2)

Ordenar por alfabético

Al seleccionar la opción de ordenar por orden alfabético utilizo el método de ordenación burbuja.

Así como mencionaba anteriormente, el método burbuja es eficiente al realizar la ordenación, claro que tiene sus ventajas y desventajas, una desventaja que encontré en el metodo de insertion sort por ejemplo, fue que no es un eficaz en tiempo cuando la lista de elementos a ordenar es muy grande.

… “desventaja es que no funciona tan bien como otros algoritmos mejores de ordenamiento. Con n al cuadrado pasos requeridos para cada n elemento a ser ordenado, este algoritmo no funciona bien con una lista grande. Por lo tanto, este sólo es útil cuando se ordena una lista de pocos elementos”.

https://techlandia.com/ordenar-lista-enlazada-java-como_47341/

for(i variable)	Accede a cada puntaje de apuesta realizada	O(n)
for(j variable)	Accede a cada puntaje -1 de apuesta realizada	O(n)
If	sí la letra actual es mayor a la letra siguiente se realiza el cambio	O(1)
Else (implícito)	No se realiza el cambio	O(1)
	Complejidad	O(n^2)

```
public Apuesta[] OrdenAlfabetico(Apuesta[] apuesta) {
    Apuesta auxiliar;

    tInicial = System.nanoTime();
    for (int i = 0; i < (apuesta.length - 1); i++) {

        for (int j = 0; j < (apuesta.length - 1); j++) {

            if (apuesta[j].getNombre().compareToIgnoreCase(apuesta[j + 1].getNombre()) > 0) { // s
                auxiliar = apuesta[j];
                apuesta[j] = apuesta[j + 1];
                apuesta[j + 1] = auxiliar;
                pasos++;
            }
        }

    }
    tFinal = System.nanoTime();
    return apuesta;
}
```

Actualización de reportes

Según los datos obtenidos de los cálculos se realiza la actualización a los reportes de la ordenación.

reporte.setTiempoOrdenamiento()	Se ingresa el dato obtenido al reporte.	O(1)
reporte.setPasosOrdenamiento ()	Se ingresa el dato obtenido al reporte.	O(1)
reporte.setMaxOrdenamiento ()	Se ingresa el dato obtenido al reporte.	O(1)
reporte.setMinOrdenamiento()	Se ingresa el dato obtenido al reporte.	O(1)
	Complejidad	O(1)

```java
public Reportes actualizarReportes(Reportes reporte){

    reporte.setTiempoOrdenamiento(getTiempoOrdenamiento());
    reporte.setPasosOrdenamientos(getPasosOrdenamiento());
    reporte.setMaxpasosOrdenamientos(0);
    reporte.setMinpasosOrdenamientos(0);

    return reporte;
}
```

Realizar la exportación de archivos y reportes

Para realizar la exportación de los reportes utilizo este método, el cual recibe como parámetro el texto que se va a escribir en el archivo y se procede a guardar como contenido la cadena de texto que se recibió.

If (comparacion)	Si en el JFileChooser se da click en continuar se crea el archivo con el texto ingresado.	O(1)
Else (implicito)	Si se da en el botón de cancelar, no se escribe en el archivo el texto recibido.	O(1)
	Complejidad	O(1)

```java
public String guardarArchivo(String lectura) {

    String path = "";
    JFileChooser ventanaSeleccionar = new JFileChooser();

    if (ventanaSeleccionar.showDialog(null, "Guardar") == JFileChooser.APPROVE_OPTION) {
        File archivo;
        archivo = ventanaSeleccionar.getSelectedFile();

        //crearArchivo(archivo.getAbsolutePath() + ".txt");
        AgregarAlArchivo(archivo.getAbsolutePath() + ".csv", lectura);
        path = archivo.getAbsolutePath();

    }
    return path;
}
```