Universidad San Carlos de Guatemala
Centro Universitario de Occidente
División de Ciencias de la Ingeniería
Estructura de Datos

Documentación Técnica

Luis Nery Cifuentes Rodas
Carné: 202030482
08/03/2022

Documentación Técnica

Método de Verificación de apuestas

```java
//Verifica las apuestas para proceder a eliminar a una que no cumpla los requisitos
public static boolean verificarApuestas(int[] temp) {//1 paso
    long inicio = System.nanoTime();
    int contadorPasos = 1;
    boolean esApuesta = true;//1 paso
    int contadorI = 1;//1 paso
    contadorPasos += 3;
    for (int i = 0; i < temp.length; i++) {//1 + 2n pasos
        if (temp[i] == contadorI) {//1 paso
            contadorI++;//1 paso
            i = -1;//1 paso
            contadorPasos += 2;
        }
        contadorPasos += 3;
    }
    if (contadorI <= 10) {// 1 paso
        contadorPasos += 2;
        Resultados.setContadorPasosverificacion(contadorPasos);
        return false;//1 paso
    }
    contadorPasos += 1;
    Resultados.setContadorPasosverificacion(contadorPasos);
    long end = System.nanoTime();
    Resultados.setTiempoVerificacion((int)(end-inicio));
    return esApuesta;// 1 paso
    //Complejidad O(n)
    //O(n) = 6n + 6  en el peor escenario
    //O(n) = 6n + 5 en el mejor escenario
}
```

- El método que se utilizó para verificar las apuestas es un ciclo for que recorre desde 0 hasta la longitud del arreglo de las posiciones de las apuestas y busca la posición de un caballo en específico, al encontrarlo el ciclo se vuelve a reiniciar.

Método para realización de los cálculos

```java
//Es el encargado de generar los resultados
public static void compararResultados(int[] resultados, Nodo temp) {//2 pasos
    long inicio = System.nanoTime();
    int contadorPasos = 3;
    for (int i = 0; i < 10; i++) {//1 + 2n pasos
        if (resultados[i] == temp.getApuesta().getOrden_caballos()[i]) {//1 paso
            temp.getApuesta().setPuntaje_obtenido(temp.getApuesta().getPuntaje_obtenido() + (10 - i));//3 pasos
            contadorPasos += 3;
        }
        contadorPasos += 3;
    }
    Resultados.setContadorPasosCalculos(contadorPasos);
    long end = System.nanoTime();
    Resultados.setTiemporCalculos((int)(end-inicio));
    //Complejidad O(n) si no se sabe el tamaño del arreglo O(1) si si se sabe
    //O(n) = 6n + 3 en el peor escenario
    //O(n) = 2n + 3 en el mejor escenario
}
```

- El algoritmo que se utilizó fue la implementación de un ciclo for que recorre desde 1 a 10 que son las 10 posiciones en las que los caballos llegaron, dentro del bucle se hacía una validación entre los dos arrays, uno de los resultados reales y el otro de las suposiciones que el apostador hacia al ingresar su apuesta, se validaron en la misma posición ya que se busca asegurar si el apostador acertó a una de las posiciones.

Metodo de Ordenamiento por Nombre y por Puntaje

```java
public static void ordenarNombre() {
    long inicio = System.nanoTime();
    Nodo pivote = primero; // 1 paso
    Nodo actual;// 1 paso
    Apuesta temp;// 1 paso
    int contadorPasos = 3;
    while (pivote != null) {// n pasos
        actual = pivote;//1 paso
        temp = pivote.getApuesta();//1 paso
        contadorPasos += 2;
        while (actual != primero && temp.getNombre_apostador().compareToIgnoreCase(actual.getAnterior().getApuesta().getNombre_aposta
            actual.setApuesta(actual.getAnterior().getApuesta());//1 paso
            actual = actual.getAnterior();//1 paso
            contadorPasos += 4;
        }
        actual.setApuesta(temp);// 1 paso
        pivote = pivote.getSiguiente();// 1 paso
        contadorPasos += 2;
    }
    long end = System.nanoTime();
    Resultados.setContadorPasosOrdenNombre(contadorPasos);
    Resultados.setTiempoNombre((int)(end-inicio));
    //Complejidad O(n^2)
    //O(n^2) = 4n^2 + 2n + 5 en el peor escenario
    //O(n^2) = 2n^2 + 2n + 5 ne el mejor escenario
}
```

```java
//Ordena los resultados por puntaje obtenido por el metodo de insercion
public static void ordenarPuntaje() {
    long inicio = System.nanoTime();
    Nodo pivote = primero;//1 paso
    Nodo actual;//1 paso
    Apuesta temp;//1 paso
    int contadorPasos = 3;
    while (pivote != null) {//n pasos
        actual = pivote;//1 pasos
        temp = pivote.getApuesta();//1 paso
        contadorPasos += 2;
        while (actual != primero && temp.getPuntaje_obtenido() > actual.getAnterior().getApuesta().getPuntaje_obtenido()) {//2n
            actual.setApuesta(actual.getAnterior().getApuesta());//1 paso
            actual = actual.getAnterior();//1 paso
            contadorPasos += 4;
        }
        actual.setApuesta(temp);//1 paso
        pivote = pivote.getSiguiente();//1 paso
        contadorPasos += 2;
    }
    long end = System.nanoTime();
    Resultados.setContadorPasosOrdenPuntos(contadorPasos);
    Resultados.setTiempoPuntos((int)(end-inicio));
    //Complejidad O(n^2)
    //O(n^2) = 4n^2 + 2n + 5 en el peor escenario
    //O(n^2) = 2n^2 + 2n + 5 ne el mejor escenario
}
```

- Se ordeno la lista doblemente enlazada a traves del metodo de insercion ya que como sabemos este tiene una complejidad de n^2, este método se aplicó porque es mucho más sencillo y conlleva menos pasos, la estrategia fue validar los atributos de los objetos e intercambiarlos según estas validaciones no tocando los nodos ya que estos solo son conectores en la lista. Se utilizó inserción ya que en el mejor de los casos es decir en que la lista ya venga ordenada éste será de una complejidad O(n).

Método para agregar una nueva apuesta

```
//insertar al final de la lista
public static void insertarApuesta(Apuesta apuesta) {//1 paso
    if (primero == null) {// 1 paso
        ultimo = new Nodo(null, null, apuesta);//6 pasos
        primero = ultimo;//1 paso
    } else {
        Nodo nuevo = new Nodo(null, ultimo, apuesta);//6 pasos
        ultimo.setSiguiente(nuevo);//1 paso
        ultimo = nuevo;// 1 paso
    }
    //Complejidad O(1)
    //O(1) = 9 pasos en el mejor escenario
    //O(1) = 10 pasos en el peor escenario
}
```

- La estrategia fue agregar las nuevas apuestas al final de la lista, el objetivo simplemente era introducir una nueva apuesta.

Metodo de eliminacion

```java
//eliminar en cualquier posicion
public static void eliminarApuesta(Nodo eliminado) {//1 paso
    if (eliminado == primero) {//Si el nodo a eliminar es el primero | 1 paso
        primero = primero.getSiguiente();// 1 paso
        if (primero != null) {// 1 paso
            primero.setAnterior(null); // 3 pasos
        } else {
            ultimo = null;//1 paso
        }
    } else if (eliminado == ultimo) {//Si el nodo a eliminar es el ultimo | 1 paso
        ultimo = ultimo.getAnterior();//1 paso
        if (ultimo != null) {//1 paso
            ultimo.setSiguiente(null);//1 paso
        } else {
            primero = null;//1 paso
        }
    } else {//Si es un nodo intermedio
        eliminado.getAnterior().setSiguiente(eliminado.getSiguiente());//3 pasos
        eliminado.getSiguiente().setAnterior(eliminado.getAnterior());// 3 pasos
    }
    //Complejidad O(1)
    //O(1) = 9 pasos en el peor escenario
    //O(1) = 5 pasos en el mejor de los casos
}
```

- El método fue hecho para poder eliminar aquellos nodos que contenían apuestas invalidas y esta conformado por condicionales que indican si el nodo es el primer nodo de la lista, si es el último o si es un nodo intermedio, el objetivo es actualizar los nodos siguientes y anteriores de manera óptima.