CÓDIGO JAVA

```
package p3;
/**
*
* @author Mariel, Diana, Yuliana
*/
public class Titulo {
  int id;
  String tipo;
  String titulo;
  String directorx;
  String reparto;
  String pais;
  String fechaN; //Fecha en que se agregó a Netflix
  String fechaE; //Fecha de estreno
  String calificacion;
  String duracion; //en minutos
  String categoria; //en cuál está listada en netflix
  String descripcion;

  public Titulo(int id, String tipo, String t, String dir, String cast,
String p, String fN, String fE, String r, String d, String cat, String
desc) {
    this.id = id;
    this.tipo = tipo;
    this.titulo = t;
    this.directorx = dir;
    this.reparto = cast;
    this.pais = p;
    this.fechaN = fN;
    this.fechaE = fE;
    this.calificacion = r;
    this.duracion = d;
    this.categoria = cat;
```

```
    this.descripcion = desc;
  }

  public String getTitulo(){
    return titulo;
  }

  //El valor clave asociado a los objetos titulos
  public int getKey(){
    return id;
  }

  public boolean compareTo(int otro){
      return this.id < otro;
  }

  public String toString(){
    return "Id: "+id+"\nTipo: "+tipo+"\nTitulo: "+titulo+"\nDirectorx:
"+directorx+"\nCast: "+reparto+"\nPais: "+pais+"\nFecha Netflix:
"+fechaN+"\nFecha Estreno: "+fechaE+"\nCalificacion:
"+calificacion+"\nDuración: "+duracion+"\nCategoria:
"+categoria+"\nDescripcion: "+descripcion;
  }

}
////////////////////////////////////////////////////////////////////////////
/*
* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-
default.txt to change this license
* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit
this template
*/
package p3;

/**
*
* @author Mariel, Diana, Yuliana
```

```
*/
public class NodoHash<T> {
    T elem;
    NodoHash<T> sig;


    //elem: cualquier tipo de dato, en este caso son titulos
    //sig: el siguiente nodo hash
    public NodoHash(T elem){
        this.elem = elem;
        this.sig = null;
    }

    public T getElem(){
        return this.elem;
    }

    public void setElem(T elem){
       this.elem=elem;
    }

    public NodoHash<T> getSig(){
        return this.sig;
    }

    public void setSig(NodoHash<T> siguiente){
        this.sig = siguiente;
    }
    //getKey?

}
////////////////////////////////////////////////////////////////////////////
package p3;
```

```
/**
*
* @author Mariel, Diana, Yuliana
*/
public class TablaHash<T> {
    int cant = 0;
    NodoHash[] tabla;

    //Se crea una tabla según el tamaño que se indica y en cada casilla se guarda un nodo hash vacío que
    //será la cabeza de las listas que se guardarán en la tabla para manejar la coliciones
    public TablaHash(int tamano){
        tabla = new NodoHash[tamano];
    }

    public NodoHash[] getTabla(){
        return tabla;
    }

    //Inserción del elemento a la tabla, usando la función de hash según multiplicación
    public void inserta(T elem, int llave){
        NodoHash n = new NodoHash(elem);
        //donde debería de ir el elemento en la tabla
        int pos = funcHashMult(llave) % tabla.length;
        NodoHash aux = tabla[pos];
        if(aux==null){
            tabla[pos]=n;
            cant++;
        } else {
            while(aux.getSig()!=null){
            aux = aux.getSig();
        }
            if(aux.getSig()==null){
                aux.setSig(n);
```

```
                cant++;
            }
        }

    }

    //insercion segun division
    public void insertaD(T elem, int llave){
        NodoHash n = new NodoHash(elem);

        //donde debería de ir el elemento en la tabla
        int pos = funcHashDiv(llave) % tabla.length;
        NodoHash aux = tabla[pos];
        if(aux==null){
            tabla[pos]=n;
            cant++;
        } else {
            while(aux.getSig()!=null){
            aux = aux.getSig();
        }
            if(aux.getSig()==null){
                aux.setSig(n);
                cant++;
            }
        }
    }

    public boolean busca(T elem, int llave){
        //Se crea un nodo auxiliar con el elemento que se desea buscar para
poder extraer el valor de hash
        //y encontrar la posición donde debería de estar el elemento.
        boolean res = false;

        //Se obtiene la posición
        int pos = funcHashMult(llave) % tabla.length;
```

```
        //Se busca el elemento en la lista almacenada en la casilla
indicada por la funcion hash
        NodoHash actual = tabla[pos].getSig();
        while(!res && actual != null){
            if(actual.getElem() == elem)
                res = true;
            actual = actual.getSig();
        }
        return res;
    }

    //Para buscar un elemento en específico en la tabla según division
    public boolean buscaD(T elem, int llave){
        //Se crea un nodo auxiliar con el elemento que se desea buscar para
poder extraer el valor de hash
        //y encontrar la posición donde debería de estar el elemento.
        boolean res = false;

        //Se obtiene la posición
        int pos = funcHashDiv(llave) % tabla.length;

        //Se busca el elemento en la lista almacenada en la casilla
indicada por la funcion hash
        NodoHash actual = tabla[pos].getSig();
        while(!res && actual != null){
            if(actual.getElem() == elem)
                res = true;
            actual = actual.getSig();
        }
        return res;
    }

    //Para borrar un elemento de la tabla hash
    public boolean borra(T elem, int llave){
        boolean res = false;
```

```
        //Se obtiene la posición en donde debería de estar el elemento
según la
        //función hash
        int pos = funcHashMult(llave) % tabla.length;
        NodoHash prev = tabla[pos];

        //Nos poscisionamos en la cabeza de la lista
        NodoHash actual = prev.getSig();

        //Buscamos la lista mientras que esta no se acabe y mientras que no
se encuentre el elemento que se quiere borrar
        while(actual != null && actual.getElem() != elem){
            prev = actual;
            actual = actual.getSig();
        }

        NodoHash aux;
        //cuando se sale del while porque se encontró un nodo igual al que
se buscaba borrar, se redirigen los atributos de sig
        //del nodo anterior para 'borrar' el actual
        if(actual != null && actual.getElem()==elem){
            aux = actual.getSig();
            prev.setSig(aux);
            cant -= 1;
            res = true;
        }

        return res;
    }

    public boolean borraD(T elem, int llave){
        boolean res = false;
        //Se obtiene la posición en donde debería de estar el elemento
según la
        //función hash
        int pos = funcHashDiv(llave) % tabla.length;
```

```
        NodoHash prev = tabla[pos];

        //Nos poscisionamos en la cabeza de la lista
        NodoHash actual = prev.getSig();

        //Buscamos la lista mientras que esta no se acabe y mientras que no se encuentre el elemento que se quiere borrar
        while(actual != null && actual.getElem() != elem){
            prev = actual;
            actual = actual.getSig();
        }

        NodoHash aux;
        //cuando se sale del while porque se encontró un nodo igual al que se buscaba borrar, se redirigen los atributos de sig
        //del nodo anterior para 'borrar' el actual
        if(actual != null && actual.getElem()==elem){
            aux = actual.getSig();
            prev.setSig(aux);
            cant -= 1;
            res = true;
        }

        return res;
    }

    //Funcion de Hash por division
    public int funcHashDiv(int llave){
        int res = llave % tabla.length;
        return res;
    }

    //funcion de Hash por multiplicacion
    public int funcHashMult(int llave){
        int aux = (int)(1/((1+Math.sqrt(5))/2));
```

```
    return aux;
}

//Regresa la cantidad de datos de una casilla en concreto
public int[] numDatosXCasilla(){
    int[] res = new int[tabla.length];
    for(int i=0; i<res.length; i++){
        int cont = 0;
        NodoHash actual = tabla[i];
        if(actual!=null){
            actual.getSig();
            while(actual != null){
            cont += 1;
            actual = actual.getSig();
        }
        }

        res[i] = cont;
    }
    return res;
}

public double promDatosXCasilla(){
    int res = 0;
    int[] cant = numDatosXCasilla();
    for(int i=0; i<cant.length; i++){
        res += cant[i];
    }
    return res/cant.length;
}

public int numcasillasVacias(){
    int cont = 0;
    for(int i=0; i<tabla.length; i++){
```

```
            NodoHash actual = tabla[i];


            if(actual==null){
                cont+=1;
            }
        }
        return cont;
    }

    public void toStr(){
        for(int i=0; i<tabla.length; i++){
            NodoHash actual = tabla[i].getSig();
            System.out.println("Casilla "+i);
            while(actual!=null){
                System.out.println(actual.getElem());
                actual = actual.getSig();
            }
        }
    }


}
////////////////////////////////////////////////////////////////////////////
package p3;
import java.io.File;
import java.io.FileNotFoundException;
import static java.lang.System.currentTimeMillis;
import java.util.ArrayList;
import java.util.Scanner;


/*
* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-
default.txt to change this license
* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit
this template
*/
```

```
/**
*
* @author Mariel, Diana, Yuliana
*/
public class Practica3 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

//      pruebaComparacion(100, 100, 8000,lecturaArchivo());
//      pruebaComparacion(1000, 1000, 8000,lecturaArchivo());
//      pruebaComparacion(5000, 5000, 8000,lecturaArchivo());
//      pruebaComparacion(100, 1000, 8000,lecturaArchivo());
//      pruebaComparacion(1000, 5000, 8000,lecturaArchivo());
//      pruebaComparacion(5000, 8000, 8000,lecturaArchivo());
//      pruebaComparacion(1000, 100, 8000,lecturaArchivo());
//      pruebaComparacion(5000, 1000, 8000,lecturaArchivo());
//      pruebaComparacion(8000, 5000, 8000,lecturaArchivo());


//      pruebasDatos(100, 100, lecturaArchivo());
//      pruebasDatos(1000, 1000, lecturaArchivo());
//      pruebasDatos(5000, 5000, lecturaArchivo());
//      pruebasDatos(100, 1000, lecturaArchivo());
//      pruebasDatos(1000, 5000, lecturaArchivo());
//      pruebasDatos(5000, 8000, lecturaArchivo());
//      pruebasDatos(1000, 100, lecturaArchivo());
//      pruebasDatos(5000, 1000, lecturaArchivo());
//      pruebasDatos(8000, 5000, lecturaArchivo());
//
```

```
    }


    public static ArrayList<Titulo> lecturaArchivo(){
      try {
        File myObj = new File("titulosNetflix.txt");
        Scanner lector = new Scanner(myObj);
        ArrayList<Titulo> l = new ArrayList<Titulo>();
        for(int i=0; i<8008; i++){
          String data = lector.nextLine();
          String[] d = data.split("/", 12);
          Titulo t = new
Titulo(Integer.parseInt(d[0]),d[1],d[2],d[3],d[4],d[5],d[6],d[7],d[8],d[9],
d[10],d[11]);
          l.add(t);
        }
      lector.close();
      return l;
      } catch (FileNotFoundException e) {
          System.out.println("Error.");
          e.printStackTrace();
          return null;
        }
    }
    //Métodos de prueba



    //Método de prueba relacionado al análisis de la cantidad de datos
almacenados en cada casilla
    //de la tabla, el número de ellas que están vacías y el promedio de la
cantidad de datos que hay en las listas.
    public static void pruebasDatos(int n, int m, ArrayList<Titulo>
listaTitulos){
        //Creación de una lista de n películas, sea n dado como parámetro
        ArrayList<Titulo> t = new ArrayList<Titulo>();
        for(int j=0; j<n;j++){
                t.add(listaTitulos.get(j));
```

```
        }


        //Creación de una tabla de hash con tamaño m, sea m dado como
parámetro
        TablaHash tabla = new TablaHash(m);


        //Se insertan las n peliculas en la tabla de hash
        for(int i=0; i<n; i++)
            tabla.insertaD(t.get(i), t.get(i).getKey());


        int[] datosXCasilla = tabla.numDatosXCasilla();
        int numVacias = tabla.numcasillasVacias();
        double promDatos = tabla.promDatosXCasilla();


        for(int i=0; i<datosXCasilla.length; i++){
            System.out.println("Hay "+datosXCasilla[i]+" datos en la
casilla "+i+" de la tabla");
        }
        System.out.println();
        System.out.println("Hay "+numVacias+" casillas vacias en la
tabla.");
        System.out.println("El promedio de datos por casilla es:
"+promDatos);


        tabla.toStr();
    }


    //Método de prueba relacionado a la comparación del desempeño de la
inserción con dos diferentes métodos de
    //hash: el método de la división y el de la multiplicación. En esta
prueba se obtienen los datos necesarios para el
    //análisis de la cantidad de datos almacenados en cada casilla de la
tabla, el número de ellas que están
    //vacías y el promedio de la cantidad de datos que hay en las listas
después de haber la inserción correspondiente.
//    public static void pruebaComparacion(int n, int m, double tope,
ArrayList<Titulo> listaTitulos){
//        double tiemp;
```

```
//         //Creación de una lista de n películas, sea n dado como
parámetro
//        ArrayList<Titulo> t = new ArrayList<Titulo>();
//        for(int j=0; j<n;j++){
//                t.add(listaTitulos.get(j));
//        }
//
//        // Se crean dos tablas de hash de tamaño m (dada como parámetro)
para realizar los
//        //dos diferentes tipos de inserción.
//        TablaHash t1 = new TablaHash(m); //metodo division
//        TablaHash t2 = new TablaHash(m); //metodo multiplicacion
//
//        ArrayList<Double> insDiv = new ArrayList<Double>();
//        ArrayList<Double> insMult = new ArrayList<Double>();
//
//        //Para t1
//        double tiempoInicio = currentTimeMillis();
//        for(int i=0; i<n; i++)
//            t1.insertaD(t.get(i), t.get(i).getKey());
//        for(int i=0; i<n;i++){
//            double tiempoFin = currentTimeMillis();
//            double tiempo = tiempoFin - tiempoInicio;
//            double p = tiempo;
//            if(p>tope)
//                tiemp = tope;
//            else
//                tiemp = p;
//            insDiv.add(tiemp);
//        }
//
//        //Para t2
//        tiempoInicio = currentTimeMillis();
//        for(int i=0; i<n; i++)
//            t2.inserta(t.get(i), t.get(i).getKey());
```

```
//        for(int i=0;i<n;i++){
//            double tiempoFin = currentTimeMillis();
//            double tiempo = tiempoFin - tiempoInicio;
//            double p = tiempo;
//            if(p>tope)
//                tiemp = tope;
//            else
//                tiemp = p;
//            insMult.add(tiemp);
//        }
//
//        int[] datosDiv = t1.numDatosXCasilla();
//        int[] datosMult = t2.numDatosXCasilla();
//        int numVaciasDiv = t1.numcasillasVacias();
//        int numVaciasMult = t2.numcasillasVacias();
//        double promDatosDiv = t1.promDatosXCasilla();
//        double promDatosMult = t2.promDatosXCasilla();
//        for(int i=0; i<datosDiv.length; i++){
//            System.out.println("Hay "+datosDiv[i]+" datos en la casilla
"+i+" de la tabla según división");
//        }
//        System.out.println();
//        for(int i=0; i<datosMult.length; i++){
//            System.out.println("Hay "+datosMult[i]+" datos en la casilla
"+i+" de la tabla según multiplicación");
//        }
//        System.out.println();
//        System.out.println("Hay "+numVaciasDiv+" casillas vacias en la
tabla según división.");
//        System.out.println();
//        System.out.println("Hay "+numVaciasMult+" casillas vacias en la
tabla según multiplicación.");
//        System.out.println();
//        System.out.println("El promedio de datos en la tabla según
división por casilla es: "+promDatosDiv);
//        System.out.println();
```

```
//        System.out.println("El promedio de datos en la tabla según
multiplicación por casilla es: "+promDatosMult);
//
//
//        double ins = 0;
//        double bus = 0;
//        double borr = 0;
//
//        //El cálculo de tiempo se va a hacer 20 veces según
multiplicación
//            double[] promIns = new double[20];
//            double[] promBus = new double[20];
//            double[] promBorr = new double[20];
//            for(int j=0; j<20; j++){
//                //Se hace una primera medición del valor de tiempo antes
de
//                //procesar los n datos del tamaño de entrada propia de la
vuelta
//                tiempoInicio = currentTimeMillis();
//                //Se hace una segunda medición del valor de tiempo
después de procesar los n datos
//                for(int k=0; k<t.size(); k++)
//                    t1.inserta(t.get(k), t.get(k).getKey());
//                double tiempoFin = currentTimeMillis();
//                //Se calcula el tiempo tomado del proceso
//                double tiempo = tiempoFin - tiempoInicio;
//                //Se inserta el tiempo en la lista de promedios
particulares.
//                promIns[j] = tiempo;
//
//                //Este procedimiento se hace para los tres procesos
considerados:
//                //insertar, buscar, borrar un dato
//                tiempoInicio = currentTimeMillis();
//                for(int k=0; k<t.size(); k++)
//                    t1.busca(t.get(k), t.get(k).getKey());
//                tiempoFin = currentTimeMillis();
```

```
//                tiempo = tiempoFin - tiempoInicio;
//                promBus[j] = tiempo;
//
//                tiempoInicio = currentTimeMillis();
//                for(int k=0; k<t.size(); k++)
//                    t1.borra(t.get(k), t.get(k).getKey());
//                tiempoFin = currentTimeMillis();
//                tiempo = tiempoFin - tiempoInicio;
//                promBorr[j] = tiempo;
//            }
//
//            //Se obtiene el promedio de cada una
//            for(int j=0; j<20; j++){
//                ins += promIns[j];
//                bus += promBus[j];
//                borr += promBorr[j];
//            }
//
//            ins = ins / 20;
//            bus = bus/20;
//            borr = borr/20;
//
//            System.out.println("Para insertar según multiplicación toma:
"+ins);
//            System.out.println("Para buscar según multiplicación toma:
"+bus);
//            System.out.println("Para borrar según multiplicación toma:
"+borr);
//            System.out.println();
//
//            ins = 0;
//        bus = 0;
//       borr = 0;
//
//        //El cálculo de tiempo se va a hacer 20 veces según
multiplicación
```

```
//           promIns = new double[20];
//            promBus = new double[20];
//            promBorr = new double[20];
//            for(int j=0; j<20; j++){
//                //Se hace una primera medición del valor de tiempo antes
de
//                //procesar los n datos del tamaño de entrada propia de la
vuelta
//                tiempoInicio = currentTimeMillis();
//                //Se hace una segunda medición del valor de tiempo
después de procesar los n datos
//                for(int k=0; k<t.size(); k++)
//                    t1.insertaD(t.get(k), t.get(k).getKey());
//                double tiempoFin = currentTimeMillis();
//                //Se calcula el tiempo tomado del proceso
//                double tiempo = tiempoFin - tiempoInicio;
//                //Se inserta el tiempo en la lista de promedios
particulares.
//                promIns[j] = tiempo;
//
//                //Este procedimiento se hace para los tres procesos
considerados:
//                //insertar, buscar, borrar un dato
//                tiempoInicio = currentTimeMillis();
//                for(int k=0; k<t.size(); k++)
//                    t1.buscaD(t.get(k), t.get(k).getKey());
//                tiempoFin = currentTimeMillis();
//                tiempo = tiempoFin - tiempoInicio;
//                promBus[j] = tiempo;
//
//                tiempoInicio = currentTimeMillis();
//                for(int k=0; k<t.size(); k++)
//                    t1.borraD(t.get(k), t.get(k).getKey());
//                tiempoFin = currentTimeMillis();
//                tiempo = tiempoFin - tiempoInicio;
//                promBorr[j] = tiempo;
```

```
//            }
//
//            //Se obtiene el promedio de cada una
//            for(int j=0; j<20; j++){
//                ins += promIns[j];
//                bus += promBus[j];
//                borr += promBorr[j];
//            }
//
//            ins = ins / 20;
//            bus = bus/20;
//            borr = borr/20;
//
//            System.out.println("Para insertar según división toma:
"+ins);
//            System.out.println("Para buscar según división toma: "+bus);
//            System.out.println("Para borrar según división toma: "+borr);
//
//    }


}
```