

Alumne: Carlos Martínez

Professor: Marc Ruiz

**ESTRUCTURES DE DADES**

**-TAULES DE HASH-**

Contenido

[Classes utilitzades 4](#_Toc106222888)

[Fitxers creats 4](#_Toc106222889)

[Aspectes a destacar 4](#_Toc106222890)

[Problemes trobats durant el desenvolupament 4](#_Toc106222891)

[TAD TAULA HASH 5](#_Toc106222892)

[Creació de les excepcions pròpies 5](#_Toc106222893)

[Mètodes destacables 6](#_Toc106222894)

[Creació de la taula 6](#_Toc106222895)

[Càlcul del hash donada una dada clau 6](#_Toc106222896)

[Assignació d’element a taula de hash 7](#_Toc106222897)

[Buscar un element a la taula 8](#_Toc106222898)

[Redimensionar la taula 8](#_Toc106222899)

[Esborrar un element de la taula de hash 9](#_Toc106222900)

[Joc de proves 10](#_Toc106222901)

[Crear llista 10](#_Toc106222902)

[Inserir elements 11](#_Toc106222903)

[Mètode buscar element 11](#_Toc106222904)

[Cèrques fetes 12](#_Toc106222905)

[Mètode redimensionar la taula 13](#_Toc106222906)

[Col·lisions 13](#_Toc106222907)

[Anàlisi del cost 14](#_Toc106222908)

[ACCESSOS: 15](#_Toc106222909)

[Llista 15](#_Toc106222910)

[Taula de hash 15](#_Toc106222911)

[STDEV: 16](#_Toc106222912)

[Llista 16](#_Toc106222913)

[Taula de hash 16](#_Toc106222914)

[Codi Font 17](#_Toc106222915)

[Taula de Hash 17](#_Toc106222916)

[Main 21](#_Toc106222917)

[Nodo 26](#_Toc106222918)

# Classes utilitzades

* Ciutadà: no era necessari implementar l’ús de la classe ciutadà pero ho fet per a fer proves de cerca amb el DNI d’un ciutadà.
* Nodo: representa cada node de la posición de la taula de hash.Guardo la clau clau, valor i hash de la dada que conté.
* HashTable: estructura principal d’aquesta segona fase de la pràctica. Conté tots els mètodes encarregats de calcular els hashes, comprovar si algún element existeix, etc.
* ListaDoble: llista doblement encadenada de la primera part de la pràctica

# Fitxers creats

Per a una visualització més simple he fet que el programa creei 3 fitxers en total:

* LogCerques: fitxers on guardo el número de cerques que s’han fet fins que s’ha pogut trobar l’element o fins que hem recorregut tots els elements sense haver-lo trobat
* hashCodes.txt: fitxer on guardo cada element de la taula de 50000 elements per a poder provar el seu funcionament i veure si un element es troba dins de la taula.

# Aspectes a destacar

## Problemes trobats durant el desenvolupament

-Problemes a la hora de redimensionar la taula ja que no recalculava el hash y un cop redimensionada si volia trobar un element, no era posible ja que havia canviat el seu hash.

-Errors amb les col·lisions ja que al principi no assignava bé les col·lisions a la posición de la taula corresponent.

-Problemes de redimensionament, a vegades redimensionava la taula innecessàriament però ho vaig acabar corregint.

## TAD TAULA HASH

Texto

Descripción generada automáticamente

## Creació de les excepcions pròpies

-ElementoNoEncontrado:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

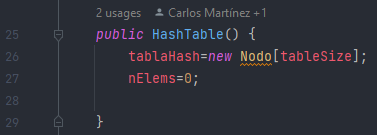
-NoSePuede:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

# Mètodes destacables

## Creació de la taula



## Càlcul del hash donada una dada clau

Calculem el hash amb el toString de l’objecte

Texto

Descripción generada automáticamente

## Assignació d’element a taula de hash

Texto

Descripción generada automáticamente

## Buscar un element a la taula

Si trobem l’element el retornem el nombre d’iteracions fetes, si no, seguim buscant.

Texto

Descripción generada automáticamente

## Redimensionar la taula

Texto

Descripción generada automáticamente

## Esborrar un element de la taula de hash

Texto

Descripción generada automáticamente

# Joc de proves

Primer de tot he creat una taula amb 10 posicions buides.

## Crear llista

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

## Inserir elements

8 ciutadansInterfaz de usuario gráfica, Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente con confianza baja

## Mètode buscar element

EXISTEIXEN: Agafem el tercer per a fer la prova

Imagen de la pantalla de una computadora

Descripción generada automáticamente con confianza baja



Texto

Descripción generada automáticamente

## Interfaz de usuario gráfica, Texto Descripción generada automáticamenteCèrques fetes

Texto

Descripción generada automáticamenteImagen que contiene Texto

Descripción generada automáticamente

50000 cerques

1000 cerques

Mètode redimensionar la taula

Captura de pantalla de computadora

Descripción generada automáticamente

Abans de redimensionar

Texto

Descripción generada automáticamente

Després de redimensionar

## Col·lisions

Buscarem els elements amb la seva clau

Interfaz de usuario gráfica

Descripción generada automáticamente

# Anàlisi del cost

Fent moltes més proves de les que he inclòs en aquest document m’he adonat del potencial de les taules de hash. La velocitat de cerca és molt superior a la que podria tenir una llista estàtica i m’ha interessat molt aquest fet.

Com només aplicant una operació podem reduir de gran forma les iteracions a fer per a poder trobar un element. A més la velocitat de cerca és extremadament ràpida com hem pogut comprovar, respecte a una cerca amb una llista doblement enllaçada com la de la primera part.

Fent un anàlisi basant-nos en moltes proves podem comprovar la gran diferencia d’accessos que hi ha entre una taula de hash i una llista a l’hora de trobar un element.

Com indica el guió de la pràctica, he fet diferents taules de hash i llistes de 1000 elements fins a 50000 elements, sumant de 1000 en 1000.

Per cada una d’aquestes taules he fet 1000 cerques d’elements generats aleatòriament i fent un promig de proves he comprovat la gran diferencia que hi ha. Faig un breu anàlisi però a l’excel estan tots els càlculs ben desenvolupats.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mida | Taula de hash | | Llista encadenada | |
| Accessos | desvest | Accessos | desvest |
| 1000 | 2,812019036 | 1,749255096 | 455,0885402 | 343,6008806 |
| 5000 | 2,716108626 | 2,292981763 | 1827,167658 | 1409,069324 |
| 10000 | 2,77880517 | 1,759373095 | 3540,456148 | 2714,474981 |
| 20000 | 4,503028618 | 2,438097533 | 6971,861079 | 5337,171837 |
| 50000 | 2,8737614 | 2,472879112 | 17149,38392 | 13090,67336 |

Veiem que les diferències són molt grans. Portant aquestes dades a una gràfica obtenim el següent:

## ACCESSOS:

### Llista

Podem veure que el cost és lineal ja que ha de recòrrer element a element.

### Taula de hash

El cost és constant i com a molt varia 1 o 2 accesos.

## STDEV:

### Llista

### Taula de hash

# Codi Font

## Taula de Hash

*package* Data;  
*import* java.io.\*;  
  
*import* Exceptions.\*;  
*public class* HashTable <K,T *extends* Comparable<T>>*implements* TADTaulaHash<K,T>{  
 *int* tableSize=10;  
 Nodo<K,T>[] tablaHash;  
 *int* nElems;  
 *public* HashTable(*int* nElems){  
 tablaHash=*new* Nodo[nElems];  
 tableSize=nElems;  
 }  
 */\*  
 \* Constructor principal de la clase HashTable  
 \*/  
 public* HashTable() {  
 Crear();  
  
  
 }  
 *public int* getIndex(K key){  
 *int* hash=hash(key);  
 *return* hash%tablaHash.length;  
 }  
  
 */\*\*  
 \* Method used to return an object's hash  
 \* @param key key from which we want to obtain the hash  
 \* @return hash of the key passed by parameter  
 \*/  
 public int* hash(K key){  
  
 String str=key.toString();  
 *int* hash=0;*//=key.hashCode();  
  
 for*(*int* i=1;i<str.length();i++){  
  
 hash+=((*int*)str.charAt(i))\*(32+i);  
 }  
 hash=hash < 0 ? hash \* -1 : hash;  
 *return* hash;  
 }  
  
 */\*\*  
 \* Alternative method used to return an object's hash  
 \* @param key object from we want to obtain the hash  
 \* @return hash of the object passed by parameter  
 \*/  
 public int* hashKey(K key){  
 String str=key.toString();  
 *int* res=0;  
 *for*(*int* i=0;i<str.length();i++){  
 res+=str.charAt(i)\*Math.pow(32,i);  
 }  
 res=res < 0 ? res \* -1 : res;  
 *return* res;  
 }  
 *@Override  
 public void* Crear() {  
 tablaHash=*new* Nodo[tableSize];  
 nElems=0;  
 }  
  
 *@Override  
 public void* Inserir(K key, T data) {  
  
 *if*(factorCarga()>=0.75){  
 *//printTable();  
 try*{  
 resize();  
 }*catch*(ElementoNoEncontrado ignored){  
 }  
 }  
 *int* hash=hash(key);  
 *int* index=getIndex(key);  
 *if*(tablaHash[index]==*null*){  
 tablaHash[index]=*new* Nodo<>(key,data,hash);  
 nElems++;  
 }  
 *else*{  
 *try*{  
 *int* offset=Buscar(key);  
 replace(index,offset,data);  
 }*catch* (ElementoNoEncontrado e){  
 tablaHash[index].add(key,data,hash);  
  
 }  
  
 }  
  
 }  
  
 *@Override  
 public* T Obtenir(K key) {  
 *return null*;  
 }  
  
 *@Override  
 public int* Buscar(K key) *throws* ElementoNoEncontrado{  
 *int* hash=hash(key);  
 *int* index=hash%tablaHash.length;  
 *int* c=0;  
 Nodo<K,T>temp=tablaHash[index];  
 *if*(temp!=*null*){  
 *while*(temp!=*null*){  
 *if*(temp.key.equals(key)){  
 *return* c;  
 }  
 temp=temp.nextCol;  
 c++;  
 }  
  
 }  
 *throw new* ElementoNoEncontrado(c);  
 }  
 *@Override  
 public int* Mida() {  
 *return* tablaHash.length;  
 }  
  
 */\*\*  
 \* Mètode que esborra un element en cas que existeixi  
 \* @param key clau de l'element  
 \* @throws ElementoNoEncontrado throws exception if element isn't found  
 \*/  
 @Override  
 public void* Esborrar(K key) *throws* ElementoNoEncontrado {  
 *int* hash=hash(key);  
 *int* index=getIndex(key);  
 Nodo<K, T> elem=tablaHash[index];  
 *if*(elem!=*null*){  
 *try* {  
 *int* offset = Buscar(key);  
 *if* (offset == 0) {  
 elem=elem.nextCol;  
 } *else* {  
 *while* (elem.nextCol.hash != hash) {  
 elem = elem.nextCol;  
 }  
 elem.nextCol = elem.nextCol.nextCol;  
 }  
 }*catch*(ElementoNoEncontrado e){  
 System.out.println(e.getMessage());  
 }  
 }  
 *else*{  
 *throw new* ElementoNoEncontrado(0);  
 }  
 }  
 *public* ListaDoble<K,T>ObtenirValors(){  
 Nodo<K,T> aux;  
 ListaDoble<K,T> listaAux=*new* ListaDoble<>();  
 *for* (Nodo<K, T> hash : tablaHash) {  
 aux = hash;  
 *while* (aux != *null*) {  
 listaAux.Inserir(aux.getData());  
 aux = aux.nextCol;  
 }  
 }  
 *return* listaAux;  
 }  
  
 *public* ListaDoble<K,T>ObtenirClaus(){  
 Nodo<K,T>aux;  
  
 ListaDoble<K,T> listaAux=*new* ListaDoble<>();  
 *for* (Nodo<K, T> hash : tablaHash) {  
 aux = hash;  
 *while* (aux != *null*) {  
  
 listaAux.Inserir((T) aux.key);  
 aux = aux.nextCol;  
 }  
 }  
 *return* listaAux;  
 }  
  
  
 */\*\*  
 \* Método que calcula el factor de carga para saber si hay que redimensionar la tabla de hash  
 \* @return true si hay que redimensionar  
 \*/  
 public float* factorCarga() {  
  
 *return* (*float*)nElems/tablaHash.length;  
 }  
 */\*\*  
 \* Método encargado de redimensionar la tabla de hash y recalcular todos los hashes de nuevo  
 \*/  
 public void* resize() *throws* ElementoNoEncontrado {  
 *//Nodo[]listaAux=new Nodo[tablaHash.length\*2];* HashTable<K,T> tablaAux=*new* HashTable<>(tablaHash.length\*2);  
 Nodo<K,T> temp;  
 K key;  
 T data;  
 *for* (Nodo<K, T> hash : tablaHash) {  
 temp = hash;  
 *while* (temp != *null*) {  
 key = temp.getKey();  
 data = temp.getData();  
 tablaAux.Inserir(key, data);  
 temp = temp.nextCol;  
 }  
  
 }  
 tablaHash=tablaAux.tablaHash;  
 tableSize=tablaHash.length;  
  
 }  
  
 */\*\*  
 \* Mètode auxiliar per a sobreescriure un valor en cas de que la clau ja existeixi  
 \* @param index index de la taula de hahs  
 \* @param offset número de la col·lisió  
 \* @param data nou valor a assignar  
 \*/  
 public void* replace(*int* index,*int* offset,T data){  
 Nodo node=tablaHash[index];  
 *int* i=0;  
 *while*(i!=offset&&node!=*null*){  
 node=node.nextCol;  
 i++;  
 }  
 *try*{  
 node.setData(data);  
  
 }*catch*(NullPointerException e){  
  
 }  
  
 }  
 */\*\*  
 \* METHOD USED TO WRITE A FILE WITH ALL THE VALUES THAT CONTAINS THE HASH TABLE  
 \*/  
 public void* writeFile () {  
  
 String fileName;  
 FileWriter escribir=*null*;  
 *int* nElems=tablaHash.length;  
 Nodo temp=*null*;  
 *try* {  
  
 fileName="hashCodes.txt";  
 escribir=*new* FileWriter(fileName);  
 }*catch*(IOException e) {  
 System.out.println(e.getMessage());  
 }  
 *// TODO Auto-generated catch block  
 for*(*int* i=0;i<nElems;i++) {  
 *try* {  
 *if*(tablaHash[i]!=*null*) {  
 temp=tablaHash[i];  
 *while*(temp!=*null*) {  
 escribir.write("key= "+i+"; "+temp.data+"; hash= "+temp.hash+"\n");  
 *//escribir.write();* escribir.flush();  
 temp=temp.nextCol;   
 }  
 }   
 }*catch*(NullPointerException e) {  
 System.out.println(e.getMessage());  
 } *catch* (IOException e) {  
 *throw new* RuntimeException(e);  
 }  
 }  
 *try* {  
 escribir.close();  
 } *catch* (IOException e) {  
 *throw new* RuntimeException(e);  
 }  
  
 }  
  
  
}

## Main

*package* Programa;  
  
*import* java.io.\*;  
*import* java.util.\*;  
  
*import* Data.\*;  
*import* Exceptions.ElementoNoEncontrado;  
  
*public class* main {  
 *public static* FileWriter file;  
 *public static* Scanner scan;  
  
 *public static void* main(String[] args)*throws* InterruptedException, IOException {  
 *// PROGRAMA PRINCIPAL PART HASHINGS* HashTable<String,Ciutada>tablaAux=*new* HashTable<>();  
  
 Ciutada carlos=*new* Ciutada("Carlos","Martinez","49424598J");  
 Ciutada david=*new* Ciutada("David","Marti","7771391023");  
 Ciutada nil=*new* Ciutada("Carlos","Martinez","44548898T");  
 Ciutada genis=*new* Ciutada("Genis","Martinez","73981391P");  
 Ciutada roger=*new* Ciutada("Roger","Massana","3731918T");  
 Ciutada lluis=*new* Ciutada("Lluis","Gallart","713739189O");  
 Ciutada gerard=*new* Ciutada("Gerard","Panisello","3241233Y");  
 Ciutada eros=*new* Ciutada("Eros","Villar","1413133T");  
  
 *try*{  
 tablaAux.Inserir("49424598J",carlos);  
 tablaAux.Inserir("7771391023",david);  
 tablaAux.Inserir("44548898T",nil);  
 tablaAux.Inserir("73981391P",genis);  
 tablaAux.Inserir("3731918T",roger);  
 tablaAux.Inserir("713739189O",lluis);  
 tablaAux.Inserir("3241233Y",gerard);  
 tablaAux.Inserir("1413133",eros);  
  
 System.out.println(tablaAux.Buscar("49424598J"));  
 System.out.println(tablaAux.Buscar("7771391023"));  
 System.out.println(tablaAux.Buscar("44548898T"));  
 System.out.println(tablaAux.Buscar("73981391P"));  
 System.out.println(tablaAux.Buscar("3731918T"));  
 System.out.println(tablaAux.Buscar("713739189O"));  
 System.out.println(tablaAux.Buscar("3241233Y"));  
 System.out.println(tablaAux.Buscar("1413133"));  
  
  
  
 }*catch*(ElementoNoEncontrado e){  
  
 }  
  
  
  
 *//Joc de proves de la taula de hash* mostrarMenu();  
 System.out.println("TABLE DONE");  
 *//Anàlisi de la llista doblement encadenada* JocProvesLlista();  
 System.out.println("LIST DONE");  
 }  
   
   
 *public static void* mostrarMenu() *throws* IOException {  
  
 *int* searchElems;  
   
 System.out.println("BENVINGUT/UDA AL PROGRAMA PRINCIPAL");  
 System.out.println("A continuació s'executaran els següents mètodes de manera automàtica:");  
 System.out.println("-Després de finalitzar cada inerció dels elements a la taula de hash anirem buscant" +  
 " números generats aleatoriament");  
 System.out.println("-Un cop generades totes les taules de hash i calculats els seus costos d'accès," +  
 "generarem un fitxer que recollirà un anàlisi del cost mig i desviació estandard tenint en compte" +  
 "els factors mencionats previament.");  
 System.out.println("COMENCEM...\n");  
 *//t.sleep(3500);  
   
   
 int* nElems=1000;  
 HashTable<Integer,Integer> numbers;  
 *int*[]digits;  
 *int* i=0;  
 ArrayList<ArrayList<Integer>>llistaAux=*new* ArrayList<>();  
 *do* {  
 System.out.println("nElems= "+nElems);  
 llistaAux.add(*new* ArrayList<>());  
 numbers=*new* HashTable<>(nElems);  
 digits = *new int*[nElems];  
  
 *for*(*int* k=0;k<nElems;k++) {  
 digits[k] = randomInt(nElems/2);  
  
 numbers.Inserir(digits[k],digits[k]);  
 }  
  
 String fileName="Analisi/LogCerques/"+nElems+"searches.txt";  
 PrintStream output = *null*;  
 *try* {  
 output = *new* PrintStream(*new* FileOutputStream(fileName));  
 } *catch* (FileNotFoundException e) {  
 *throw new* RuntimeException(e);  
 }  
 System.setOut(output);  
 *//Escribimos en un fichero los elementos que encuentra y los que no, para poder visualizarlo más comodamente  
  
 for*(*int* j=0;j<nElems;j++) {  
  
 *try*{  
 searchElems=numbers.Buscar(randomInt(numbers.Mida()/2));  
 llistaAux.get(i).add(searchElems);  
 *//totalSearches[i]+=searchElems;* System.out.println(searchElems+" iteration until element has been found");  
  
 }*catch*(ElementoNoEncontrado e){  
 System.out.println(e.getMessage());  
 }  
  
 }  
 System.setOut(*new* PrintStream(*new* FileOutputStream(FileDescriptor.out)));  
 nElems+=1000;i++;  
 }*while*(nElems<=50000);  
 numbers.writeFile();  
 scan=*new* Scanner(System.in);  
 System.out.println("Introduce el numero a buscar");  
 *int* n=scan.nextInt();  
 *try*{  
 System.out.println(numbers.Buscar(n));  
  
 }*catch*(ElementoNoEncontrado e){  
  
 }  
 *//Escritura del fitxer* FileWriter analisis=*new* FileWriter("Analisi/CostCompuTaula.csv");  
 analisis.write("MIDA;"+"N ACCESSOS;"+"DESV EST\n");  
 *int* Elems=1000;  
 *for* (ArrayList<Integer> lista : llistaAux) {  
 analisis.write(Elems + ";" + mean(lista) + ";" + stDev(lista)+"\n");  
 Elems += 1000;  
 }  
 analisis.close();  
  
  
 }  
 *public static void* JocProvesLlista(){  
 ListaDoble<Integer,Integer>lista=*new* ListaDoble<>();  
 *int* nElems=1000;  
 *int*[]digits;  
 *int* searchElems=0;  
 ArrayList<ArrayList<Integer>>llistaAux=*new* ArrayList<>();  
 *int* i=0;  
 FileWriter analisis=*null*;  
 *try* {  
 analisis = *new* FileWriter("Analisi/CostCompuLlista.csv");  
 analisis.write("MIDA;"+"N ACCESSOS;"+"DESV EST\n");  
 }*catch*(IOException e) {  
 }  
 *do*{  
 lista=*new* ListaDoble<>();  
 System.out.println("nElems= "+nElems);  
 llistaAux.add(*new* ArrayList<>());  
 digits=*new int*[nElems];  
 *for*(*int* j=0; j<nElems; j++){  
 digits[j]=randomInt(nElems/2);  
 lista.Inserir(digits[j]);  
 }  
 *for*(*int* k=0;k<nElems;k++){  
 *try*{  
 searchElems=lista.Buscar(randomInt(nElems/2));  
 llistaAux.get(i).add(searchElems);  
 }*catch*(ElementoNoEncontrado e){  
 *//System.out.println(e.getMessage());* }  
 }  
  
  
 *try* {  
  
 analisis.write(nElems + ";" + mean(llistaAux.get(i)) + ";" + stDev(llistaAux.get(i)) + "\n");  
 }  
 *catch*(NullPointerException e){  
 System.out.println(e.getMessage());  
 } *catch* (IOException e) {  
 System.out.println("FILE ERROR");  
 }  
 i++;  
 nElems+=1000;  
 }*while*(nElems<=50000);  
 *try*{  
 analisis.close();  
  
 }*catch*(IOException e){  
  
 }  
  
  
 }  
 *public static double* mean(ArrayList<Integer> lista){  
 *int* sum=0;  
 *for*(*int* temp: lista){  
 sum+=temp;  
 }  
 *return* (*double*)sum/ lista.size();  
  
 }  
 *public static double* stDev(ArrayList<Integer>lista) {  
 *int*[]nums=*new int*[lista.size()];  
 *for*(*int* i=0;i<lista.size();i++){  
 nums[i]=lista.get(i);  
 }  
 *double* stDev=0.0,sum=0.0;  
 *for*(*int* i=0;i<nums.length&&nums[i]!=0;i++) {  
 sum+=nums[i];  
 }  
 *double* media=sum/nums.length;  
 *for*(*int* j=0;j<nums.length&&nums[j]!=0;j++) {  
 stDev+=Math.pow(nums[j]-media, 2);  
 }  
 *double* sq = stDev / nums.length;  
 stDev = Math.sqrt(sq);  
 *return* stDev;  
 }  
   
 *public static void* separator() {  
 System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 }  
  
 */\*\*  
 \* Método que se encarga de pedir un elemento y buscarlo en la lista pasada como parámetro  
 \* @param table tabla en la que buscaremos el elemento que se introduzca  
 \*/  
 public static void* searchElement(HashTable table) {  
 String data= "";  
 *do* {  
 System.out.println("Escriu l'element que vulguis buscar: ");  
 scan=*new* Scanner(System.in);  
 *try* {  
 data=scan.next();  
 *//else* System.out.println(table.Buscar(data));  
 }*catch*(NumberFormatException e){  
 System.out.println("EL número introduit no és vàlid");  
 }  
 *catch*(ElementoNoEncontrado e){  
 System.out.println(e.getMessage());  
 }  
 }*while*(!data.equalsIgnoreCase("-1"));  
  
 }  
  
  
  
 */\*\*  
 \* Genera un número entero aleatorio  
 \* @return random int  
 \*/  
 public static int* randomInt(*int* rightLimit) {  
 *int* leftLimit=1;  
 *//int rightLimit;  
 int* number;  
  
 *//rightLimit=Integer.MAX\_VALUE;  
 //rightLimit=(leftLimit\*10)-1;* number=leftLimit+(*int*)(Math.random()\*(rightLimit-leftLimit));  
 *return* number;  
 }  
   
}

## Nodo

*package* Data;  
  
*public class* Nodo<K,T *extends* Comparable<T>> *implements* Comparable<T> {  
 T data;  
 K key;  
 *public* Nodo<K,T> nextCol;  
 *public* Nodo<K,T> prev;  
 *int* hash=0;  
  
 *public* Nodo(Nodo<K,T> col,T data) {  
 *this*.data=data;  
 nextCol=col;  
 prev=*null*;  
 }  
 *public* Nodo(K key,T data,*int* hash){  
 *this*.data=data;  
 *this*.key=key;  
 *this*.hash=hash;  
 nextCol=*null*;  
 prev=*null*;  
 }  
  
 *public* Nodo(Nodo<K,T> sig, Nodo<K,T> prev, T data) {  
 *this*.data = data;  
 *this*.nextCol = sig;  
 *this*.prev = prev;  
 }  
  
 *public* Nodo(T data2) {  
 *this*(*null*,data2);  
 }  
 *public void* add(K key,T data,*int* hash){  
 Nodo<K,T> temp=*this*;  
 *if*(temp.data==*null*){  
 temp.data=data;  
 temp.hash=hash;  
 }  
 *else* {  
 *while* (temp.nextCol != *null*) {  
 temp = temp.nextCol;  
 }  
 temp.setNextCol(*new* Nodo<>(key,data,hash));  
 }  
 }  
  
 *public* K getKey() {  
 *return* key;  
 }  
  
 *public* T getData() {  
 *return this*.data;  
 }  
 *public void* setData(T data) {  
 *this*.data=data;  
 }  
  
 *public void* setNextCol(Nodo<K,T> col) {  
 *this*.nextCol=col;  
 }  
  
 *@Override  
 public* String toString() {  
 *return* "Nodo{" +  
 "data=" + data +  
 ", nextCol=" + nextCol +  
 '}';  
 }  
 *@Override  
 public int* hashCode() {  
 *return* hash;  
 }  
  
 *@Override  
 public int* compareTo(T o) {  
 *if*(o.hashCode()==hashCode())*return* 0;  
 *return* -1;  
 }  
}