

**Estructura de Dades**

**Pràctica 2: Grafs**

**curs 2022-23**

**Estudiant:** Matías Larrosa

**Professor/a:** Marc Ruiz

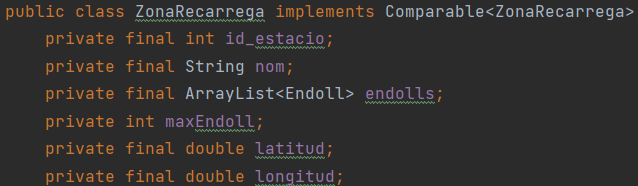
**Data de lliurament:** 29/05/2022

# Decisions de disseny

L’estructura que s’ha implementat per el graf es la millora de la multillista d’adjacència que només guarda una relació entre els dos nodes estalviant així espai.

## ZonaRecarrega

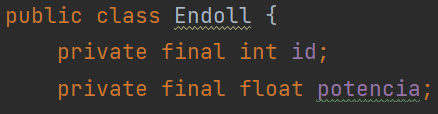
Aquesta es la classe que guarda la informació de cada node.



Guardo la id de cada estació, que es unica per a cadascuna.

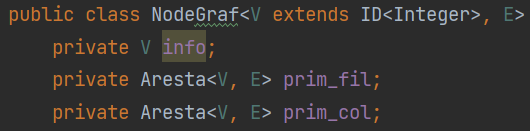
El nom de l’estació els endolls que hi ha en aquella estació, guardant l’index potencia màxima d’aquests.

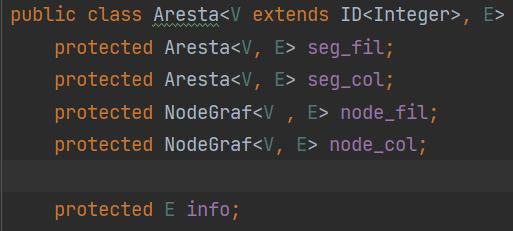
Les coordenades de la Zona de Recarrega latitud, longitud.



Endoll només guarda un identificador del endoll i el valor de la seva potencia.

## NodeGraf i Aresta

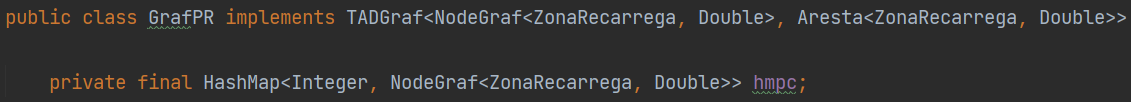




NodeGraf guardarà la informació de les zones de recarrega del graf i Aresta guardarà la informació de la relació entre 2 nodes del graf, una referencia a cadascun d’aquests nodes i a les següents arestes.

## GrafPR

És la classe principal de la practica que guarda la informació extreta de icaen.json.



### crear()

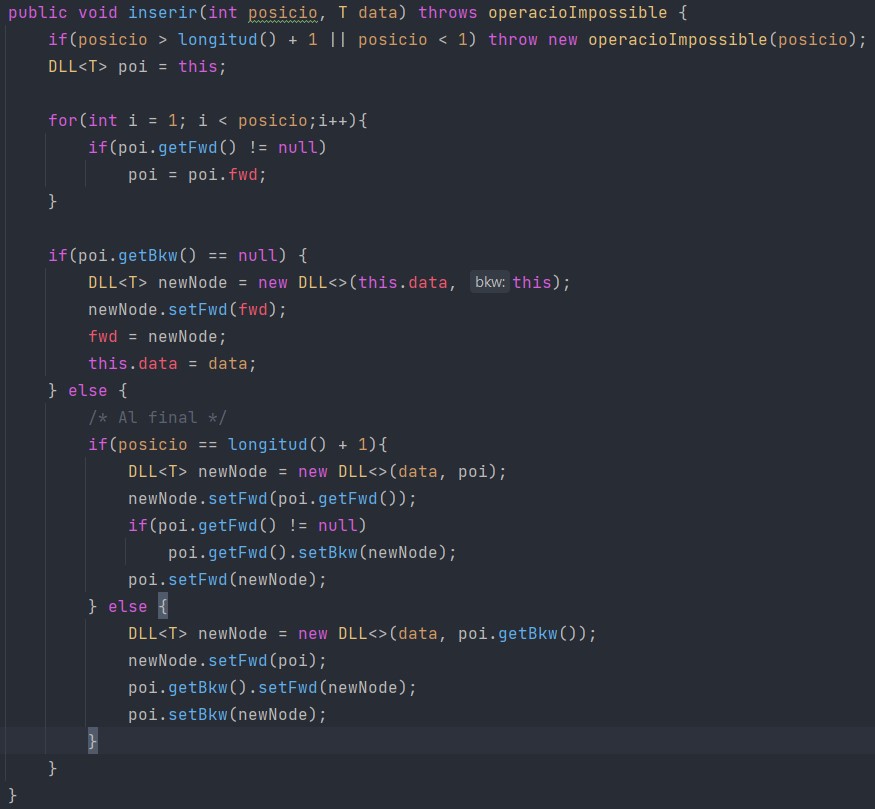
Afegeix totes les zones de recarrega, diferenciades per id\_estacio, al hashMap. I crida a enllacarNodes() que afegirà les arestes necessàries entre els nodes. Al enllaçar nodes si no s’ha pogut enllaçar un node amb cap altre ja que la distancia era superior a 40 km s’enllaçarà amb el que tingui la distancia mínima.

### afegirAresta(NodeGraf, NodeGraf, Aresta)

Aquest mètode afegirà la referencia de la aresta als dos nodes on correspongui (fil o columna) i guardara una referencia dels nodes a la aresta. Sempre guardaré la referencia de la aresta en les files si el node es més petit o en les columnes si es mes gran.

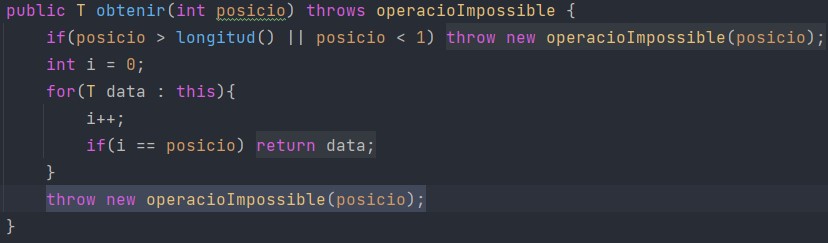
### existeixAresta(NodeGraf, NodeGraf)

Existeix aresta recorre les files del primer node, si el primer és més petit que el segon, o les columnes, si el primer es més gran que el segon. Buscant en les arestes la referencia a la columna o fila respectivament de el segon graf.



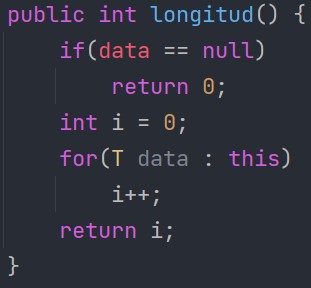
### obtenir(int)

Obtenir retorna l’element a la posició demanada. Si la posició esta fora del rang, en aquest cas [1, longitud()], es llençarà una excepció en temps de execució. Per recorre la llista i trobar la posició demanada la recorro utilitzant un for-each.



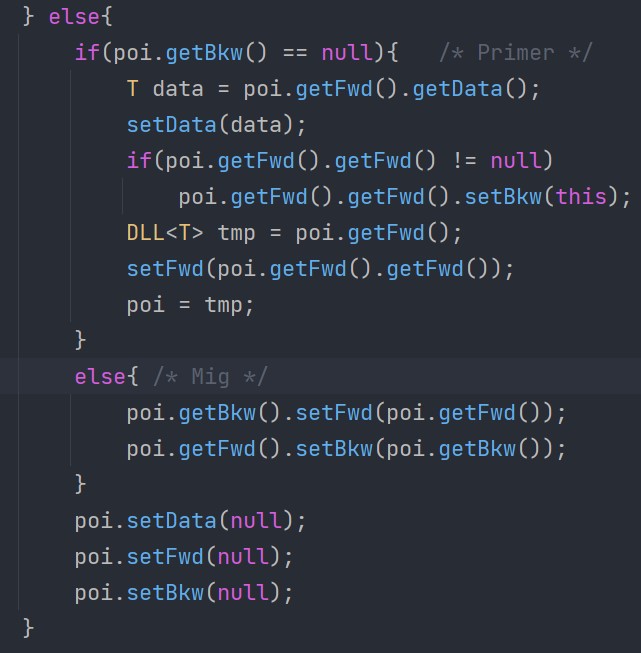
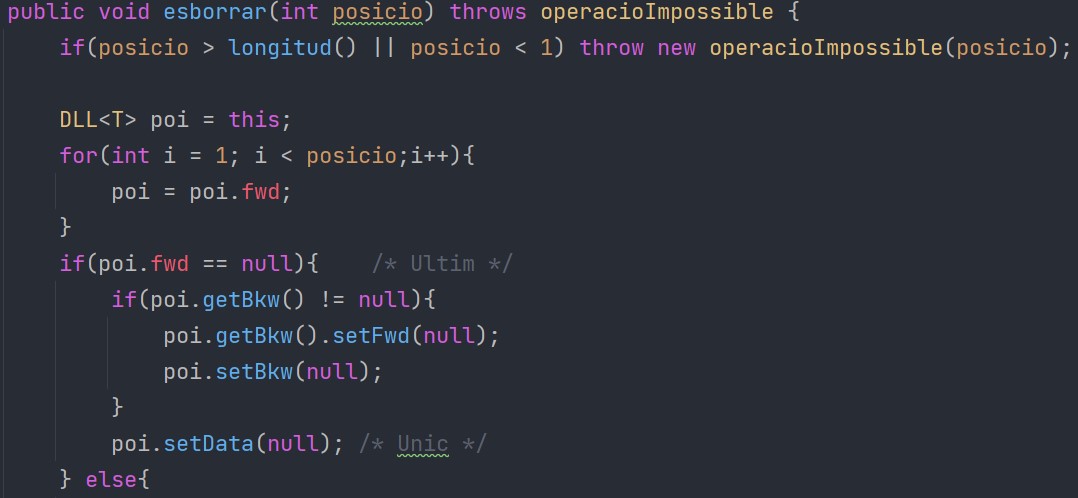
### longitud()

Retorna la longitud del la llista, utilitzant un bucle for-each per recorre-la. Es podria heber obtat per guardar la longitud de la llista i actualitzarla segons s’afegeixen o s’eliminen elements.



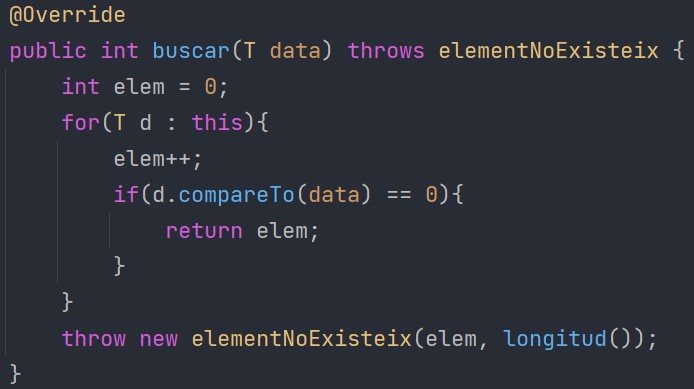
### esborrar(int)

Esborra la posició demanada sempre que estigui dins del rang correcte [1, longitud()]. Per esborrar s’ha de tenir en compte quin cas s’està tractant; si es tracta de l’últim element (fwd == null), si es l’únic element, si es el primer element o si es un del mig.



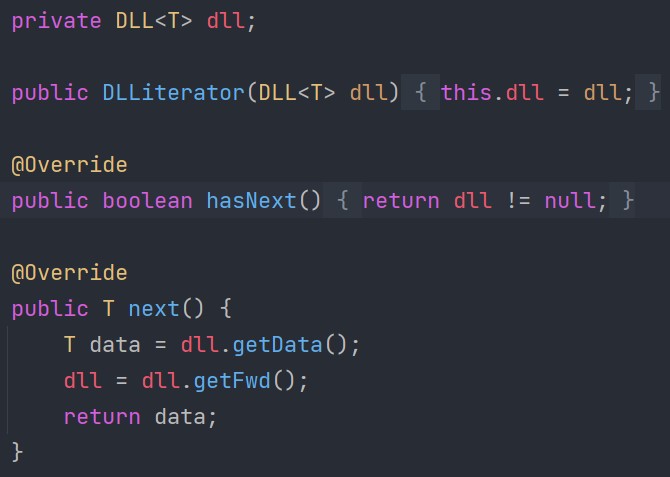
### buscar(T)

Retornarà el nombre d’elements que s’han agut de comprovar per trobar o no trobar l’element. Recorro la llista amb un bucle for-each.



## DLLIterator

Aquesta es la classe que es fa servir per poder iterar sobre la llista. Es guarda una referencia de la llista sencera que s’inicialitza al constructor se si hi ha un següent element si la referencia no està apuntant a null i per apuntar al següent només cal fer dll = dll.getFwd(). Aquest és un dels avantatges de no tenir nodes.

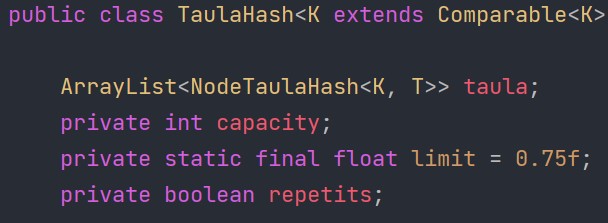


## Joc de proves DLL

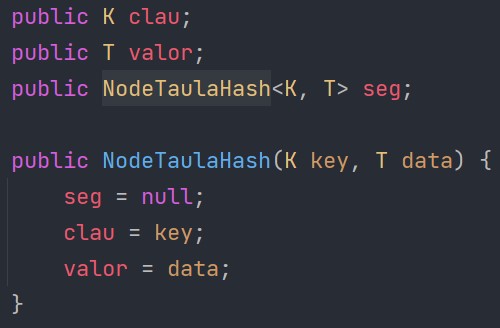
Per fer el joc de proves he fet servir la llibreria de JUnit per fer més simple el testeig i la comprovació. He deixat un main preparat per si es vol fer comprovacions extres.

## Taula de hash (TaulaHash)

La taula de hash si que utilitza un node extern a la seva estructura i emmagatzema aquestos nodes en un ArrayList inicialitzat a null en totes les seves posicions (tamany = capacity). També té un valor constant anomenat límit que correspon al valor de tall per saber si s’ha de redimensionar l’array o no. I el booleà repetits existeix per saber si es permeteixen repetits o no (a la especificació posava que si hi ha una clau repetida s’actualitza el seu valor però això crea un problema a l’hora de fer el càlcul del cost computacional).



|  |
| --- |
| + getters & setters |

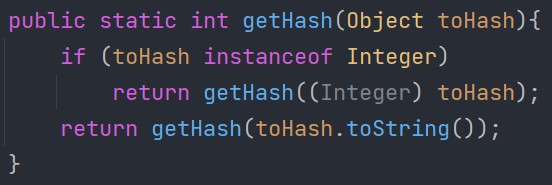


## Hasher

Hasher es la classe (amb mètode estàtics) que s’encarrega de transformar cualsevol input en un valor de hash. Per als enters retorna l’enter directament que seria lo mateix a dir hash(k) = k si k es un integer. Per a les cadenes de caràcters aplica una XOR amb el valor numèric de cada caràcter de forma alternativa (mètode de les permutacions) i per els objectes crida al toString() de l’objecte i retorna el hash de l’String.

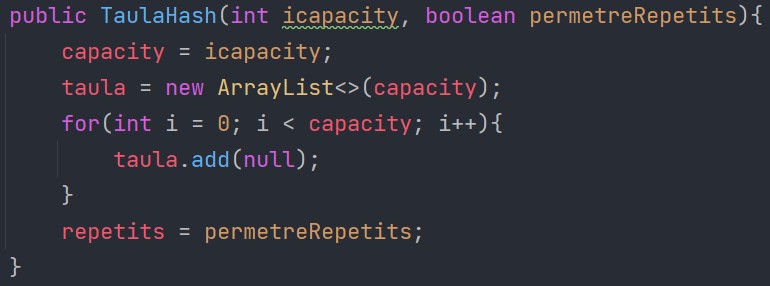
Un dels problemes d’aquest mètode es la repetició de secuencies, al estar aplicant una XOR de cada caràcter si només hi ha un 1 la XOR de 0 i cualsevol altre cosa es aquesta cosa. Per tant, hash(“a”) = ‘a’ (97) si hi ha dos “a” hash(“aa”) = 0, ja que, la XOR de dos coses iguals es 0. Però en el cas de que n’hi hagin 3 “a” o cualsevol nombre imparell la hash(“aaa”) = “a” (97) = hash(“a”).

Encara així, he triat aquest mètode dels oferits a la teoria ja que hem donava els millors resultats per a claus com Patata | patata. Però no, te en compte claus com Matias Larrosa | Larrosa Matias.



### crear()

Crear inizialitza l’array a null amb una capacitat inicial de 10 i no admet repetits. El constructor base crida a crear i, en canvi, el complexe que modifica la capacitat inicial i si es permeten repetits no el crida.

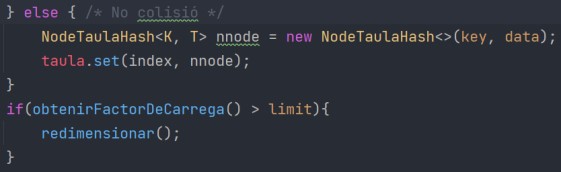


### inserir(K, T)

Inserir demana una clau i l’objecte al cual li pertany aquesta clau. Doncs l’index corresponent s’obté del hash % capacitat de la taula. Si aquesta no esta utilitzada s’assigna el valor de la posició a la data (T) i, si ho està significa que s’ha donat una colisió i per tant, cal afegir-lo al final si no existeix la mateixa clau o si es permeteixen repetits o, sino si existeix la clau es modificaran les dades.

El metode només llençara una excepció si les dades eren les mateixes a les d’entrada i no s’acceptaven repetits.

Per acabar, si el factor de carrega supera el limit caldrà redimensionar la taula.



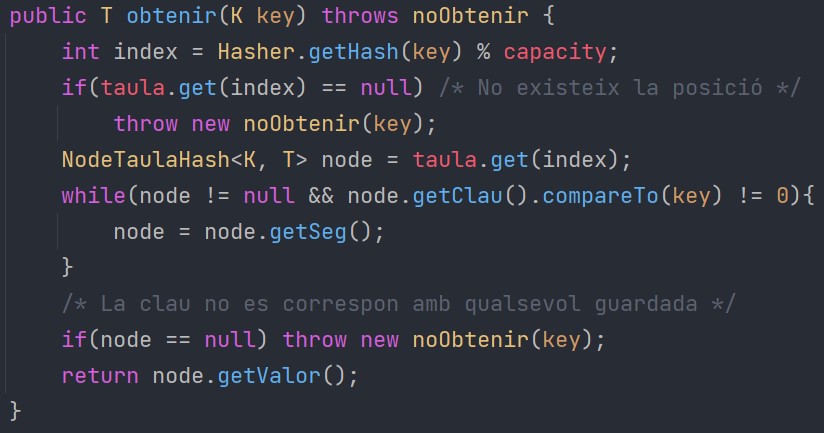
### redimensionar()

Redimensionar és un metode privat que redimensiona l’array amb una capacitat de 1.5 vegades més gran. Per fer-ho cal obtenir tots els elements i afegir-los a les posibles noves posicions.



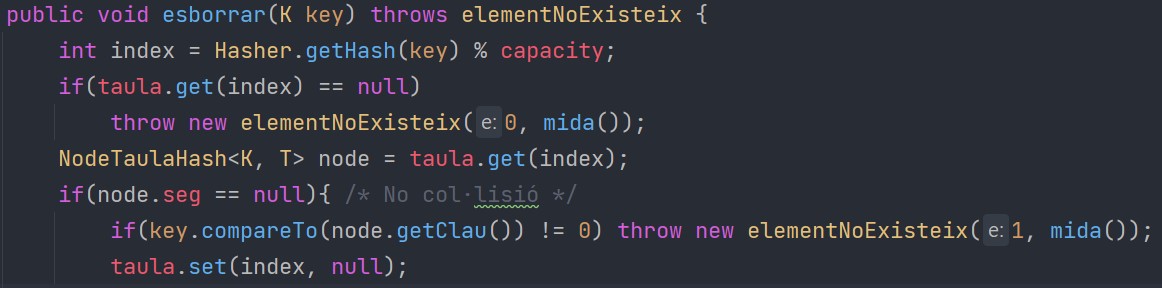
### obtenir(K)

Obtenir retorna el valor de una clau determina només si existeix la seva posició del hash i si aquest posició te la mateixa clau passada.



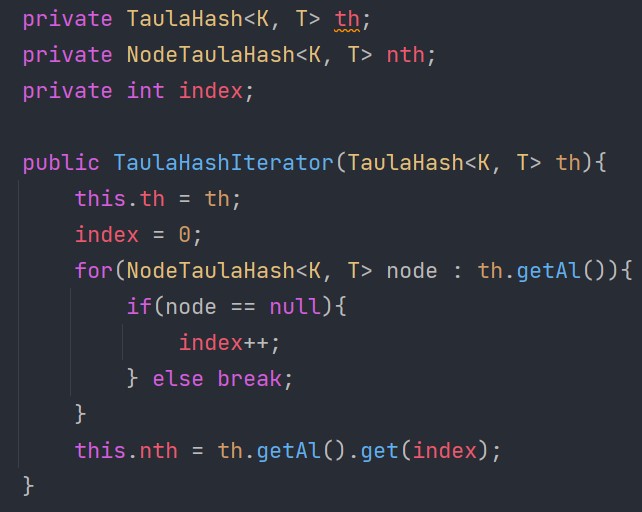
### esborrar(K)

Esborrar elimina un valor que tingui aquesta clau. Cal tenir en compte en quina posició i situació es troba aquest valor (col·lisió no col·lisió, inici, final o mig de la col·lisió). Aquest metode llençarà una excepció si la clau no es troba a la taula.



## TaulaHashIterator

Aquesta es la classe que es fa servir per iterar sobre la taula de hash. Emmagatzema una referencia a la taula de hash, un node per recorre en cas de que n’hi hagin col·lisions i un índex per saber en quina posició del arrayList estic. Se que hi han més elements si l’índex es més petit que la capacitat de la taula de hash. I per retornar el següent retorno el node actual i l’avanço al següent, si hi ha alguna col·lisió incremento l’índex fins a trobar un altre element o fins a acabar. L’índex l’inicialitzo a la primera posició on hi hagi un valor.



# Joc de proves Taula de Hash

Per fer el joc de proves he fet servir la llibreria de JUnit per fer més simple el testeig i la comprovació. He deixat un main preparat per si es vol fer comprovacions extres.

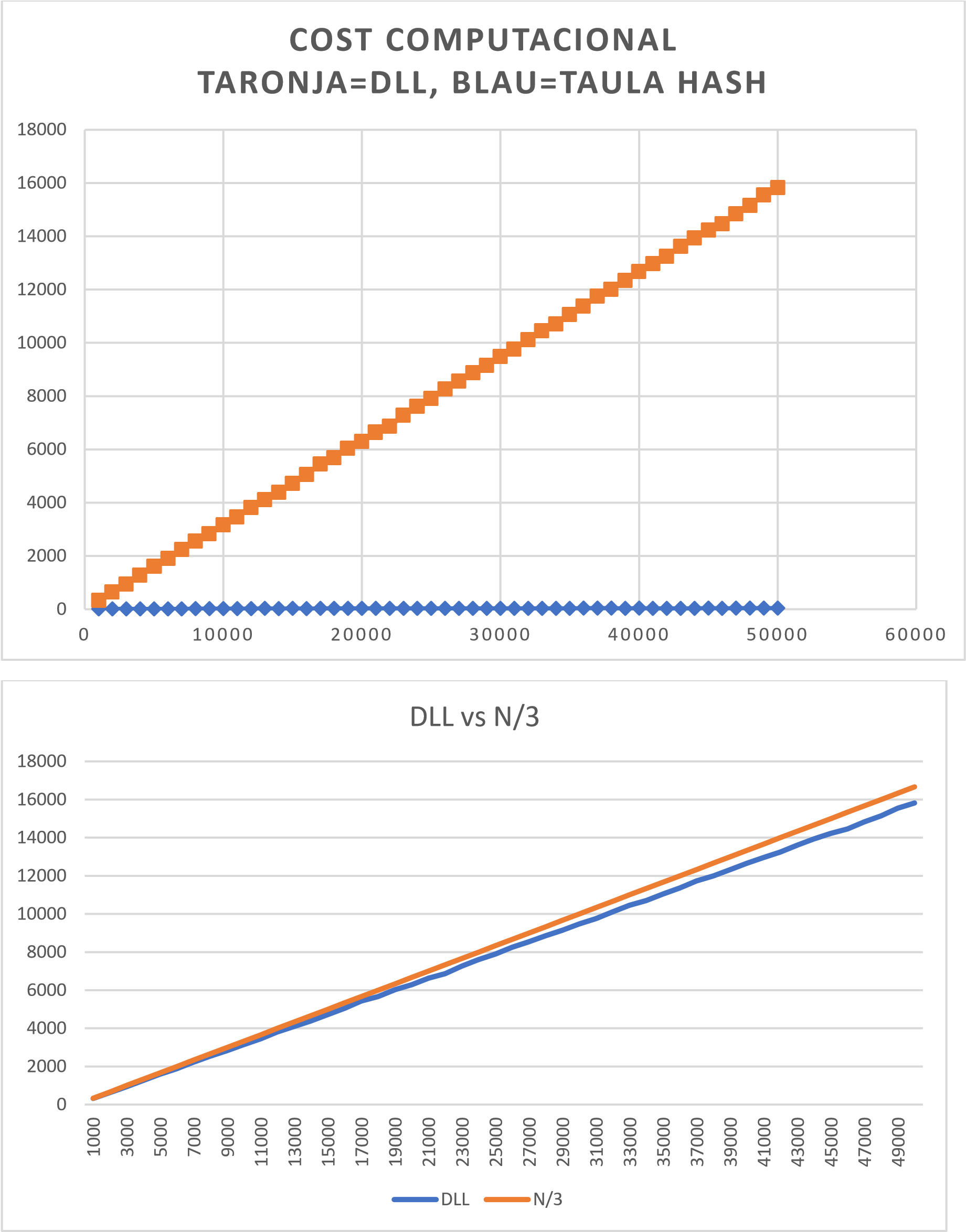
# Cost Computacional

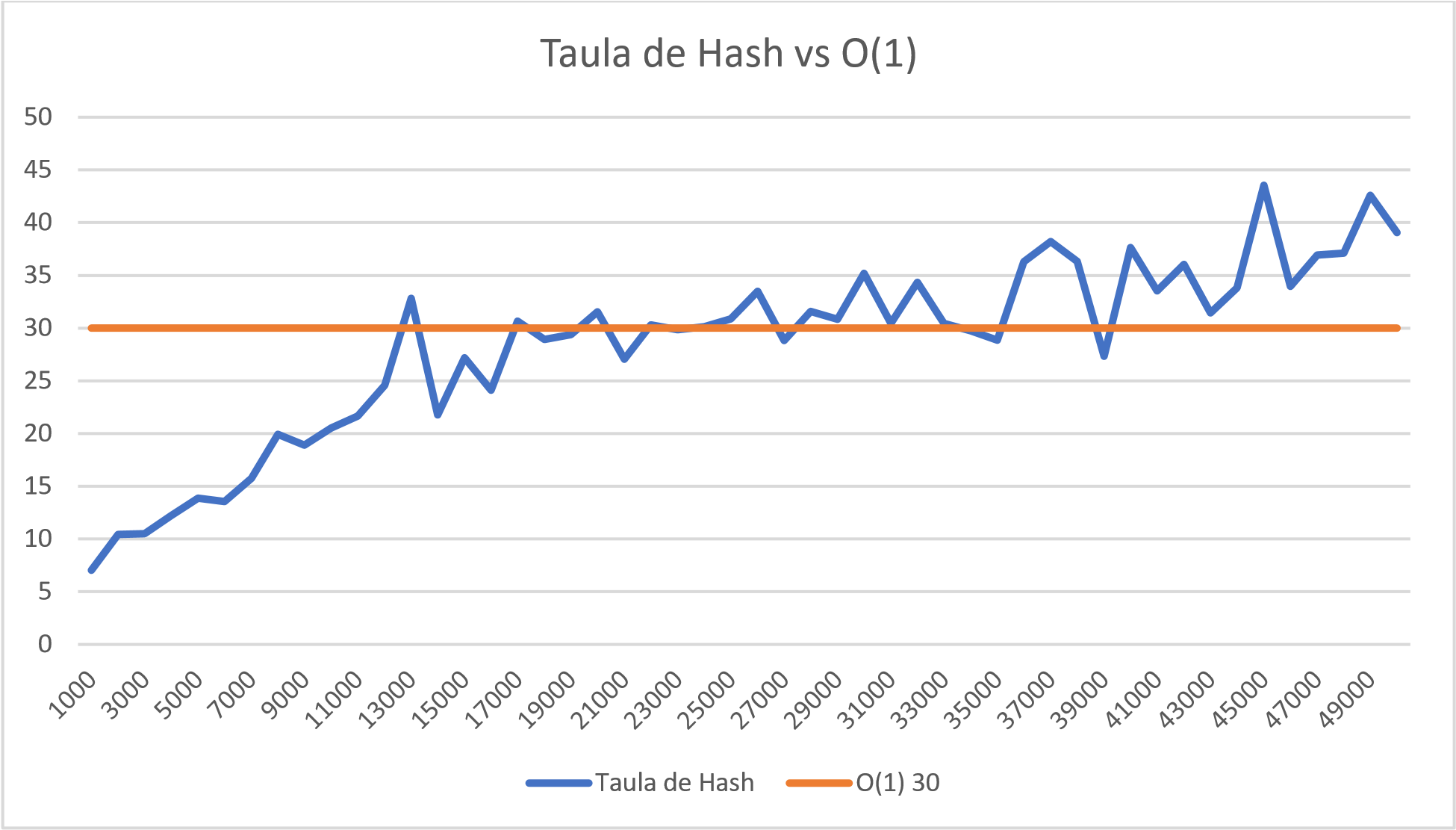
Com a part opcional de aquesta practica és demana fer un anàlisi comparatiu de l’operació buscar, aquesta, retorna el número de elements que s’han agut de recorre per comprovar l’existència o no existència d’un element. Hi ha un problema ocasionat per dos motius principals:

* A la especificació de inserir diu “Si l’element ja existia, actualitza el seu valor”, això porta a que en una llista de N elements ([1000,2000,..,49000,50000]), on es guardaran valors aleatoris de 1-N/2, en la taula de hash es guardaran com a màxim N/2 elements. Aquesta taula llavors, sempre tindira menys de la meitat que la llista doblement enllaçada (DLL).
* El meu mètode de Hash per a enters retorna el mateix nombre al ser una funció lo suficientment exhaustiva no veia inconvenient a fer servir aquest mètode.

Per tant, la taula de hash que utilitzaré en aquest anàlisi es compondra de <String, Integer> onstring serà l’enter en forma de cadena. I permetré la repetició de valors per a tenir la taula amb N elements.

La classe que fa aquest anàlisi i crea el .csv s’anomena CostTest i està al Package CostComputacional.





La taula de hash te un cost de cerca gairebé constant ja que gracies als codis de hash i han poques col·lisions, si hagués utilitzat el mètode de hash dels int i afegit elements aleatoris de l’1 – N l’acces a la busqueda hauria sigut 1 ja que no hi haurien col·lision.

També m’he percatat que la DLL te un accés de busqueda de O(n) = n/3.

# Codi Fase1 TADCiutada

|  |
| --- |
| package fase1.EstructuraDades;    public abstract class TADCiutada implements Comparable<TADCiutada> { protected String nom; protected String cognom; protected String dni;    public TADCiutada(String nom, String cognom, String dni) { this.nom = nom; this.cognom = cognom; this.dni = dni;  } public String getDni() { return dni;  }    @Override  public int compareTo(TADCiutada o) { return dni.compareTo(o.getDni());  } |

public String getNom(){

return nom;

}

@Override public String toString() {

return nom + ";" + cognom + "(" + dni + ")";

}

}

# TADLlista

|  |
| --- |
| package fase1.EstructuraDades;      public abstract class TADCiutada implements Comparable<TADCiutada> { protected String nom; protected String cognom; protected String dni;    public TADCiutada(String nom, String cognom, String dni) { this.nom = nom; this.cognom = cognom; this.dni = dni;  }  public String getDni() { return dni;  }    @Override  public int compareTo(TADCiutada o) { return dni.compareTo(o.getDni());  }  public String getNom(){ return nom;  }    @Override  public String toString() {  return nom + ";" + cognom + "(" + dni + ")";  }  } |

# ElementNoExisteix

|  |
| --- |
| package fase1.Excepcions;  public class elementNoExisteix extends Exception{ private int elements;  public elementNoExisteix(int e, int s){  super("S'han recorregut " + e + " elements a la llista de " + s + " elements i no s'ha trobat l'element."); elements = e;  } public int getElements() { return elements;  }  } |

# OperacioImpossible

|  |
| --- |
| package fase1.Excepcions;    public class operacioImpossible extends Exception{ public operacioImpossible(int p){  super("Error: no es posible fer aquesta operació. Posició " + p + " fora del rang.");    }  } |

# DLLTest

|  |
| --- |
| package fase1.JUnit;    import fase1.EstructuraDades.CiutadaPeu; import fase1.EstructuraDades.DLL; import fase1.EstructuraDades.TADCiutada; import fase1.Excepcions.elementNoExisteix; import fase1.Excepcions.operacioImpossible; import org.junit.jupiter.api.Test;    import static org.junit.jupiter.api.Assertions.\*; class DLLTest {  DLL<TADCiutada> dllc;    CiutadaPeu a = new CiutadaPeu("Matias","Larrosa Babio","a");  CiutadaPeu b = new CiutadaPeu("Nancy","Babio Sanchez","b");  CiutadaPeu c = new CiutadaPeu("Jordi","Cojuhar Cojuhar","c");  CiutadaPeu d = new CiutadaPeu("Hugo","Acedo Coronado","d");  CiutadaPeu e = new CiutadaPeu("Lucas","Larrosa Babio","e");  CiutadaPeu f = new CiutadaPeu("Marcelo","Larrosa Bermudéz","f");  CiutadaPeu[] lcp = new CiutadaPeu[]{a, b, c, d, e, f};  /\*\*   * Es comprovara si la dllc creada te els atributs correctes.   \*/ @Test void crear() { dllc = new DLL<>(); dllc.crear();  assertNotNull(dllc, "La DLL es null");  assertNull(dllc.getData(), "La data inicial creada es null"); }    /\*\*   * Es comprovara si s'han afegit els elements en l'ordre correcte.   \*/ @Test  void inserir() { dllc = new DLL<>(); dllc.inserir(a); dllc.inserir(b); dllc.inserir(c); dllc.inserir(d); dllc.inserir(e); dllc.inserir(f);    DLL<TADCiutada> poi = dllc; |

for(int i = 0;i < 6;i++,poi = poi.getFwd()){

|  |
| --- |
| assertEquals(0, poi.getData().compareTo(lcp[i]), poi + " != " + lcp[i]); }  }    /\*\*   * Es comprovara si es poden afegir elements a posicions critiques de la llista. * La primera i una del mig(4), per afegir elements al final de la llista s'ha * d'utilitzar el metode DLL.inserir(T). I, que llençi l'excepció operacioImpossible * quan la poscició introduida estigui fora del rang ([1, longitud()]).   \*/ @Test  void testInserir() { reiniciarDLL();  CiutadaPeu cp1 = new CiutadaPeu("1","1", "1");  CiutadaPeu cp2 = new CiutadaPeu("2","2", "2"); CiutadaPeu cp3 = new CiutadaPeu("3","3", "3");    assertDoesNotThrow(() -> dllc.inserir(1, cp1), "El métode inserir(int, T) ha llençat l'excepció operacioImpossible");  /\* Inserim a la primera posicio de la llista [1, a, b, c, ..., f ] \*/ assertDoesNotThrow(() -> dllc.inserir(4, cp2), "El métode inserir(int, T) ha llençat l'excepció operacioImpossible");  assertThrows(operacioImpossible.class, () -> dllc.inserir(-1, cp3), "El métode inserir(int, T) NO ha llençat l'excepcio");  assertThrows(operacioImpossible.class, () -> dllc.inserir(100, cp3), "El métode inserir(int, T) NO ha llençat l'excepcio");    assertEquals(0, dllc.getData().compareTo(cp1), "El métode inserir(int, T) en la primera posició no guarda l'element en la primera posició");  assertEquals(0, dllc.getFwd().getFwd().getFwd().getData().compareTo(cp2), "El métode inserir(int, T) en la cuarta posició no guarda l'element en la primera posició");  }    /\*\*   * Es comprovara que cada posició de el DLL utilitzant el metode obtenir(int) * retorni els objectes correctes. I, que llençi l'excepció operacioImpossible * quan la poscició introduida estigui fora del rang ([1, longitud()]).   \*/ @Test  void obtenir() { reiniciarDLL(); assertDoesNotThrow(()->{ for(int i = 0; i < 6;i++){  TADCiutada tmp = dllc.obtenir(i + 1);  assertNotNull(tmp, "L'element obtingut a la posició + " + i + 1 +  " es null");  assertEquals(0, tmp.compareTo(lcp[i]), "L'element obtingut a la posició " + i + " no es l'esperat");  }  });  assertThrows(operacioImpossible.class, () -> dllc.obtenir(100), "El métode inserir(int, T) NO ha llençat l'excepcio");  assertThrows(operacioImpossible.class, () -> dllc.obtenir(-1 ), "El |

|  |
| --- |
| métode inserir(int, T) NO ha llençat l'excepcio");  }    /\*\*   * Es comprovara que s'eliminin correctament els elements de cada posició critica * la primera, cualsevol del mig i l'ultima. I, que llençi l'excepció operacioImpossible * quan la poscició introduida estigui fora del rang ([1, longitud()]).   \*  \*/ @Test  void esborrar() { reiniciarDLL();  assertDoesNotThrow(()->{ for(int i = 0; i < 6;i++){ reiniciarDLL(); dllc.esborrar(i+1); TADCiutada tmp = lcp[i];  assertThrows(elementNoExisteix.class, ()->dllc.buscar(tmp),  "L'element " + tmp + " no s'ha eliminat correctament.");  }  });    /\* Esborrar elements de dll particulars \*/ dllc = new DLL<>(); dllc.inserir(a);  assertDoesNotThrow(()->dllc.esborrar(1)); assertNotNull(dllc); /\* Posició fora del rang \*/ reiniciarDLL();  assertThrows(operacioImpossible.class, () -> dllc.obtenir(100), "El métode inserir(int, T) NO ha llençat l'excepcio");  assertThrows(operacioImpossible.class, () -> dllc.obtenir(-1 ), "El métode inserir(int, T) NO ha llençat l'excepcio");  }    /\*\*   * Es comprovarà que la longitud comenci en 0 i, augmenti * i disminueixi a mesura que s'insereixen i s'esborren elements.   \*/ @Test  void longitud() { dllc = new DLL<>();  assertEquals(0, dllc.longitud(), "La longitud de un DLL nou ha de ser 0."  + dllc);  dllc.inserir(a);  assertEquals(1, dllc.longitud(), "La longitud de un DLL amb 1 element ha de ser 1. " + dllc ); dllc.inserir(b);  assertEquals(2, dllc.longitud(), "La longitud de un DLL amb 2 element ha de ser 2. " + dllc); dllc.inserir(c);  assertEquals(3, dllc.longitud(), "La longitud de un DLL amb 3 element ha de ser 3. " + dllc); dllc.inserir(d);  assertEquals(4, dllc.longitud(), "La longitud de un DLL amb 4 element ha de ser 4. " + dllc); dllc.inserir(e);  assertEquals(5, dllc.longitud(), "La longitud de un DLL amb 5 element ha |

de ser 5. " + dllc); dllc.inserir(f);

assertEquals(6, dllc.longitud(), "La longitud de un DLL amb 6 element ha

|  |
| --- |
| de ser 6. " + dllc);  assertDoesNotThrow(()->{ dllc.esborrar(1);  assertEquals(5, dllc.longitud(), "La longitud de un DLL amb 5 element ha de ser 5. " + dllc); dllc.esborrar(1);  assertEquals(4, dllc.longitud(), "La longitud de un DLL amb 4 element ha de ser 4. " + dllc); dllc.esborrar(1);  assertEquals(3, dllc.longitud(), "La longitud de un DLL amb 3 element ha de ser 3. " + dllc); dllc.esborrar(1);  assertEquals(2, dllc.longitud(), "La longitud de un DLL amb 2 element ha de ser 2. " + dllc); dllc.esborrar(1);  assertEquals(1, dllc.longitud(), "La longitud de un DLL amb 1 element ha de ser 1. " + dllc); dllc.esborrar(1);  assertEquals(0, dllc.longitud(), "La longitud de un DLL amb 0 element ha de ser 0. " + dllc);  }, "El mètode esborrar ha llençat l'excepció operacioImpossible");  }    /\*\*   * Es comprovara que per a cada entrada de buscar(T) el resultat sera * igual a la mateixa posició d'aquell objecte. I, que llençi l'excepció * elementNoExisteix si l'element no existeix al DLL.   \*/ @Test void buscar() { reiniciarDLL(); assertDoesNotThrow(()->{  for (int i = 0; i < dllc.longitud();i++) assertEquals(dllc.buscar(lcp[i]), i + 1,  "Per trobar l'element " + lcp[i] + " no s'han recorregut el nombre correcte de elements");  }, "El métode DLL.buscar(T) ha llençat l'excepció elementNoExisteix."); assertThrows(elementNoExisteix.class, ()->dllc.buscar(new  CiutadaPeu("element","No","Existeix")), "El métode DLL.buscar NO ha llençat l'excepció noExisteixElement. ");  }    /\*\*  \* Es comprovara que la llista iterada retorna els elements ordenadament.  \*/ @Test  void iterator() { reiniciarDLL(); int i = 0;  for(TADCiutada c : dllc){  assertEquals(0, c.compareTo(lcp[i]), "L'element retornat per l'iterator no es el correcte. " + c + " != " + lcp[i]); i++; |

}

}

private void reiniciarDLL(){ dllc = new DLL<>(); dllc.inserir(a);

dllc.inserir(b); dllc.inserir(c); dllc.inserir(d); dllc.inserir(e); dllc.inserir(f);

}

}

# DLL

|  |
| --- |
| package fase1.EstructuraDades;  import fase1.Excepcions.elementNoExisteix; import fase1.Excepcions.operacioImpossible;    import java.util.Iterator;    public class DLL<T extends Comparable<T>> implements TADLlista<T>, Iterable<T> {  private DLL<T> fwd; private DLL<T> bkw; private T data;  public DLL(){ this(null);  }  public DLL(T data){ crear();  this.data = data;  }  public DLL(T data, DLL<T> bkw){ crear(); this.data = data; this.bkw = bkw;  }    @Override  public void crear() { fwd = null; bkw = null; data = null;  }    @Override  public void inserir(T data) { if(data != null){  if (this.data != null) { DLL<T> poi = this; while(poi.getFwd() != null){ poi = poi.getFwd();  }  poi.setFwd(new DLL<>(data, poi));  } else  this.data = data;  } |

}

/\*\*

|  |
| --- |
| * Posició -> [1, longitud() + 1] * @param posicio posició a inserir element * @param data element a inserir. * @throws operacioImpossible posició fora del rang de la llista   \*/ @Override  public void inserir(int posicio, T data) throws operacioImpossible { if(posicio > longitud() + 1 || posicio < 1) throw new operacioImpossible(posicio);  DLL<T> poi = this;  for(int i = 1; i < posicio;i++){ if(poi.getFwd() != null) poi = poi.fwd;  }    if(poi.getBkw() == null) {  DLL<T> newNode = new DLL<>(this.data, this); newNode.setFwd(fwd); fwd = newNode; this.data = data;  } else {  /\* Al final \*/  if(posicio == longitud() + 1){  DLL<T> newNode = new DLL<>(data, poi); newNode.setFwd(poi.getFwd()); if(poi.getFwd() != null) poi.getFwd().setBkw(newNode); poi.setFwd(newNode);  } else {  DLL<T> newNode = new DLL<>(data, poi.getBkw()); newNode.setFwd(poi); poi.getBkw().setFwd(newNode); poi.setBkw(newNode);  }  }  }  private void setBkw(DLL<T> newNode) { bkw = newNode;  }  private void setFwd(DLL<T> fwd) { this.fwd = fwd;  }    /\*\*   * Posició -> [1, longitud()] * @param posicio posició de l'element * @return * @throws operacioImpossible posició fora del rang de la llista   \*/ @Override  public T obtenir(int posicio) throws operacioImpossible { if(posicio > longitud() || posicio < 1) throw new |

operacioImpossible(posicio); int i = 0; for(T data : this){

|  |
| --- |
| i++;  if(i == posicio) return data;  }  throw new operacioImpossible(posicio);  }    @Override  public int longitud() { if(data == null) return 0; int i = 0; for(T data : this) i++; return i;  }    /\*\*   * Posició -> [1, longitud()] * @param posicio posició de l'element a esborrar * @throws operacioImpossible posició fora del rang de la llista   \*/ @Override  public void esborrar(int posicio) throws operacioImpossible { if(posicio > longitud() || posicio < 1) throw new operacioImpossible(posicio);    DLL<T> poi = this;  for(int i = 1; i < posicio;i++){ poi = poi.fwd;  }  if(poi.fwd == null){ /\* Ultim \*/ if(poi.getBkw() != null){ poi.getBkw().setFwd(null); poi.setBkw(null);  }  poi.setData(null); /\* Unic \*/  } else{  if(poi.getBkw() == null){ /\* Primer \*/ T data = poi.getFwd().getData(); setData(data);  if(poi.getFwd().getFwd() != null) poi.getFwd().getFwd().setBkw(this); DLL<T> tmp = poi.getFwd(); setFwd(poi.getFwd().getFwd()); poi = tmp;  }  else{ /\* Mig \*/  poi.getBkw().setFwd(poi.getFwd()); poi.getFwd().setBkw(poi.getBkw());  }  poi.setData(null); poi.setFwd(null); poi.setBkw(null);  }    }    @Override  public int buscar(T data) throws elementNoExisteix { int elem = 0; for(T d : this){ |

elem++;

|  |
| --- |
| if(d.compareTo(data) == 0){ return elem;  } }  throw new elementNoExisteix(elem, longitud()); }    @Override  public Iterator<T> iterator() { return new DLLiterator<>(this);  } public T getData() { return data;  }  private void setData(T data) { this.data = data;  }  public DLL<T> getFwd() { return fwd;  }  public DLL<T> getBkw() { return bkw;  } public String toString(){ String r = "[{null"; for(T data : this){ r += "} ⇄ {" + data;  } return r + "} ⇄ {null}]";  }  } |

# DLLIterator

|  |
| --- |
| package fase1.EstructuraDades;  import java.util.Iterator;    public class DLLiterator<T extends Comparable<T>> implements Iterator<T> {    private DLL<T> dll;  public DLLiterator(DLL<T> dll){ this.dll = dll;  }    @Override  public boolean hasNext() { return dll != null;  } |

@Override public T next() { T data = dll.getData(); dll = dll.getFwd(); return data;

}

}

# CiutadaPeu

|  |
| --- |
| package fase1.EstructuraDades;    public class CiutadaPeu extends TADCiutada implements Comparable<TADCiutada>{ public CiutadaPeu(String nom, String cognom, String dni) { super(nom, cognom, dni);  }    @Override  public int compareTo(TADCiutada o) { return super.compareTo(o);  }  public void caminar(){  System.out.println("Estic caminant");  }    } |

# Fase2 Hasher

|  |
| --- |
| package fase2.EstructuraDades;    public class Hasher {    //Overload  public static int getHash(Object toHash){ if (toHash instanceof Integer) return getHash((Integer) toHash); return getHash(toHash.toString());  }    //Overload  public static int getHash(String toHash){ int result = 0;  //toHash = toHash.replaceAll(" ", "").trim(); char[] chars = new char[toHash.length()]; toHash.getChars(0, toHash.length(), chars, 0);    for(int i = 0;i < toHash.length(); i++){ result = result^(int) chars[i];  }  return result;  }    //Overload  public static int getHash(int toHash){ return toHash;  } |

}

# NodeTaulaHash

|  |
| --- |
| package fase2.EstructuraDades;    public class NodeTaulaHash<K extends Comparable<K>, T extends Comparable<T>> { public K clau; public T valor;  public NodeTaulaHash<K, T> seg;    public NodeTaulaHash(K key, T data) { seg = null; clau = key; valor = data;  }  public NodeTaulaHash() { clau = null; valor = null; seg = null;  } public K getClau() { return clau;  }  public void setClau(K clau) { this.clau = clau;  } public T getValor() { return valor;  }  public void setValor(T valor) { this.valor = valor;  }  public NodeTaulaHash<K, T> getSeg() { return seg;  }  public void setSeg(NodeTaulaHash<K, T> seg) { this.seg = seg;  }  } |

# TADHash

|  |
| --- |
| package fase2.EstructuraDades;    import fase1.EstructuraDades.DLL; import fase1.Excepcions.elementNoExisteix; import fase2.Excepcions.noInsercio; import fase2.Excepcions.noObtenir; import fase2.Excepcions.noTrobat;    public interface TADHash<K extends Comparable<K>, T extends Comparable<T>> { |

/\*\*

* Constructor per inicialitzar la taula.

\*/ void crear(); /\*\*

* Mètode per tal d'inserir un element a la taula de Hash.

|  |
| --- |
| * Si l'element ja existia s'actualitza el seu valor. Es * llença una excepció si no es pot inserir. * @param key Identificador * @param data Data * @throws noInsercio No s'ha pogut inserir.   \*/  void inserir(K key, T data) throws noInsercio;    /\*\*   * Mètode que retorna l'element que té la clau K. * @param key Identificador * @return data * @throws noObtenir No s'ha trobat key   \*/  T obtenir(K key) throws noObtenir;    /\*\*   * Mètode que comprova si un element està a la taula. * @param key Identificador * @return El coste de l'operació. Nombre d'elements que s'hagin accedit per tal de comprovar si l'element existeix o no. * @throws noTrobat No s'ha trobat l'element.   \*/  int buscar(K key) throws noTrobat;    /\*\*   * Mètode per saber la mida de la taula. * @return Retorna el nombre d'elements que conté la taula en aquest moment.   \*/  int mida();    /\*\*   * Mètode per tal d'esborrar un element de la taula. * @param key Identificador   \*/  void esborrar(K key) throws elementNoExisteix;  /\*\*   * Mètode per obtenir tots els valors. * @return Retorna una llista amb tots els valors de la taula.   \*/  DLL<T> obtenirValors();    /\*\*   * Mètode per obtenir totes les claus. * @return Retorna una llista amb totes les claus de la taula.   \*/  DLL<K> obtenirClaus();    /\*\*   * Mètode per obtenir el factor de càrrega * @return Retorna el factor de càrrega.   \*/  float obtenirFactorDeCarrega(); |

}

# TaulaHash

|  |
| --- |
| package fase2.EstructuraDades;    import fase1.EstructuraDades.DLL; import fase1.Excepcions.elementNoExisteix; import fase2.Excepcions.noInsercio; import fase2.Excepcions.noObtenir; import fase2.Excepcions.noTrobat;  import java.util.ArrayList; import java.util.Iterator;    public class TaulaHash<K extends Comparable<K>, T extends Comparable<T>> implements TADHash<K, T>, Iterable<T>{    ArrayList<NodeTaulaHash<K, T>> taula; private int capacity;  private static final float limit = 0.75f; private boolean repetits;    public TaulaHash(){ crear();  }  public TaulaHash(int icapacity, boolean permetreRepetits){ capacity = icapacity;  taula = new ArrayList<>(capacity); for(int i = 0; i < capacity; i++){ taula.add(null);  }  repetits = permetreRepetits;  }    @Override  public void crear() { capacity = 10;  taula = new ArrayList<>(capacity); for(int i = 0; i < capacity; i++){ taula.add(null);  }  repetits = false;  }    @Override  public void inserir(K key, T data) throws noInsercio { int index = Hasher.getHash(key) % capacity; if(taula.get(index) != null){ /\* Colisió \*/ NodeTaulaHash<K, T> node = taula.get(index); while(node.seg != null){  if(repetits || key.compareTo(node.getClau()) != 0) node = node.seg; else break;  }  if(repetits || key.compareTo(node.getClau()) != 0){  NodeTaulaHash<K, T> nnode = new NodeTaulaHash<>(key, data); node.setSeg(nnode);  }else {  if(node.getValor().compareTo(data) == 0) throw new noInsercio(index); node.setValor(data); |

}

} else { /\* No colisió \*/

NodeTaulaHash<K, T> nnode = new NodeTaulaHash<>(key, data);

|  |
| --- |
| taula.set(index, nnode);  }  if(obtenirFactorDeCarrega() > limit){ redimensionar();  }  }  private void redimensionar() {  capacity \*= 1.5;  ArrayList<NodeTaulaHash<K,T>> nouarray = new ArrayList<>(capacity); for(int i = 0; i < capacity; i++){ nouarray.add(null);  }  for(NodeTaulaHash<K, T> elem : taula){ while(elem != null){  int index = Hasher.getHash(elem.getClau()) % capacity;  NodeTaulaHash<K, T> nnode = new  NodeTaulaHash<>(elem.getClau(), elem.getValor());  NodeTaulaHash<K, T> node = nouarray.get(index); if(node == null){  nouarray.set(index, nnode);  } else {  while(node.getSeg() != null){ node = node.getSeg();  }  node.setSeg(nnode);  }  elem = elem.getSeg();  }  }  taula = nouarray;  }    @Override  public T obtenir(K key) throws noObtenir { int index = Hasher.getHash(key) % capacity;  if(taula.get(index) == null) /\* No existeix la posició \*/ throw new noObtenir(key);  NodeTaulaHash<K, T> node = taula.get(index);  while(node != null && node.getClau().compareTo(key) != 0){ node = node.getSeg();  }  /\* La clau no es correspon amb qualsevol guardada \*/ if(node == null) throw new noObtenir(key); return node.getValor();  }    @Override  public int buscar(K key) throws noTrobat { int collisions = 0;  int index = Hasher.getHash(key) % capacity; NodeTaulaHash<K, T> node = taula.get(index);  while(node != null && node.getClau().compareTo(key) != 0){ node = node.getSeg(); collisions++;  }  if(node == null) throw new noTrobat(collisions + 1); |

return collisions + 1;

}

@Override

|  |
| --- |
| public int mida() { int elem = 0;  for(NodeTaulaHash<K, T> n : taula){ if(n != null){ while(n != null){ elem++;  n = n.getSeg();  }  } }  return elem;  }    @Override  public void esborrar(K key) throws elementNoExisteix { int index = Hasher.getHash(key) % capacity; if(taula.get(index) == null)  throw new elementNoExisteix(0, mida()); NodeTaulaHash<K, T> node = taula.get(index); if(node.seg == null){ /\* No col·lisió \*/  if(key.compareTo(node.getClau()) != 0) throw new elementNoExisteix(1, mida());  taula.set(index, null);  } else { /\* Col·lisió \*/  NodeTaulaHash<K, T> ant = taula.get(index); int i = 0;  if(node.getClau().compareTo(key) != 0){ node = node.getSeg();  while(node != null && node.getClau().compareTo(key) != 0){ node = node.getSeg(); ant = ant.getSeg(); i++; } i++; }  if(node == null)  throw new elementNoExisteix(1 + i, mida()); if(ant == node){  taula.set(index, node.getSeg());  } else {  ant.setSeg(node.getSeg());  }  }  }    @Override  public DLL<T> obtenirValors() { DLL<T> valors = new DLL<>(); for(NodeTaulaHash<K, T> elem : taula){ if(elem != null) {  while (elem.getSeg() != null) { valors.inserir(elem.getValor()); elem = elem.getSeg();  }  valors.inserir(elem.getValor());  } }  return valors; |

}

@Override

|  |
| --- |
| public DLL<K> obtenirClaus() { DLL<K> claus = new DLL<>();  for(NodeTaulaHash<K, T> elem : taula){ if(elem != null) {  while (elem.getSeg() != null) { claus.inserir(elem.getClau()); elem = elem.getSeg();  }  claus.inserir(elem.getClau());  } }  return claus;  }    @Override  public float obtenirFactorDeCarrega() { return (float) mida()/capacity;  }  public int getCapacity(){return capacity;}    public String toString(){  StringBuilder sb = new StringBuilder(); for(NodeTaulaHash<K, T> elem : taula){ if(elem != null) {  sb.append("Index(").append(Hasher.getHash(elem.getClau()) % capacity).append("): ");  while (elem.getSeg() != null) {  sb.append("[").append(elem.getClau()).append(",  ").append(elem.getValor()).append("] -> "); elem = elem.getSeg();  }  sb.append("[").append(elem.getClau()).append(",  ").append(elem.getValor()).append("] -> "); sb.append("null\n");  } }  sb.append("Factor carrega: ").append(obtenirFactorDeCarrega()); return sb.toString();  }    @Override  public Iterator<T> iterator() {  return new TaulaHashIterator<K, T>(this); } public ArrayList<NodeTaulaHash<K,T>> getAl() { return taula;  }  } |

# TaulaHashIterator

package fase2.EstructuraDades;

import java.util.ArrayList; import java.util.Iterator;

public class TaulaHashIterator<K extends Comparable<K>, T extends Comparable<T>>

|  |
| --- |
| implements Iterator<T> {  private TaulaHash<K, T> th; private NodeTaulaHash<K, T> nth; private int index;    public TaulaHashIterator(TaulaHash<K, T> th){ this.th = th; index = 0;  for(NodeTaulaHash<K, T> node : th.getAl()){ if(node == null){ index++; } else break;  }  this.nth = th.getAl().get(index);  }    @Override  public boolean hasNext() {  return index < th.getCapacity();  }    @Override  public T next() {  ArrayList<NodeTaulaHash<K, T>> al = th.getAl();  T data = nth.getValor(); nth = nth.getSeg(); if(nth == null){  for(index = index + 1; index < th.getCapacity(); index++){ if(al.get(index) != null){ nth = al.get(index); break;  }  } }  return data;  }  } |

# NoInsercio

|  |
| --- |
| package fase2.Excepcions;    public class noInsercio extends Exception{ public noInsercio(int n){  super("L'element " + n + " no s'ha pogut inserir.");  }  } |

# NoObtenir

|  |
| --- |
| package fase2.Excepcions;    public class noObtenir extends Exception{ public noObtenir(Object n){  super("No s'ha pogut obtenir l'element de la clau " + n + ".");  }  } |

# NoTrobat

|  |
| --- |
| package fase2.Excepcions;    public class noTrobat extends Exception { private int n;    public noTrobat(int n){  super("No s'ha trobat l'element buscat, s'han buscat " + n + " elements."); this.n = n;  }  public int getN(){return n;} } |

# TaulaHashJUnit

|  |
| --- |
| package fase2.JUnit;    import fase1.EstructuraDades.CiutadaPeu; import fase1.EstructuraDades.DLL; import fase1.EstructuraDades.TADCiutada; import fase1.Excepcions.elementNoExisteix; import fase2.EstructuraDades.TaulaHash; import fase2.Excepcions.noInsercio; import fase2.Excepcions.noObtenir; import fase2.Excepcions.noTrobat; import org.junit.jupiter.api.Test;  import static org.junit.jupiter.api.Assertions.\*;    public class TaulaHashJUnit {    TaulaHash<String, TADCiutada> th;    CiutadaPeu a = new CiutadaPeu("Matias","Larrosa Babio","a");  CiutadaPeu b = new CiutadaPeu("Nancy","Babio Sanchez","b");  CiutadaPeu c = new CiutadaPeu("Jordi","Cojuhar Cojuhar","c");  CiutadaPeu d = new CiutadaPeu("Hugo","Acedo Coronado","d");  CiutadaPeu e = new CiutadaPeu("Lucas","Larrosa Babio","e");  CiutadaPeu f = new CiutadaPeu("Marcelo","Larrosa Bermudéz","f");  CiutadaPeu[] lcp = new CiutadaPeu[]{a, b, c, d, e, f};  /\*\*   * Es comprovara si la taula de hash creada te els atributs correctes   \*/ @Test void crear(){  th = new TaulaHash<>();  /\* TaulaHash() crida a crear() \*/ assertNotNull(th, "La TH es null");  assertEquals(0, th.mida(), "Hi han valors inicials en la TH que s'acaba de crear.");  }    /\*\*   * Es comprovara que s'afegeixen elements correctament.   \*/  @Test |

void inserir() throws noInsercio { th = new TaulaHash<>();

assertDoesNotThrow(() -> th.inserir(a.getDni(), a), "El metode inserir()

|  |
| --- |
| ha llençat l'excepcio noInsercio");  assertEquals(1, th.mida(), "S'ha afegit 1 element i la mida no es correspon.");  assertDoesNotThrow(() -> th.inserir(b.getDni(), b), "El metode inserir() ha llençat l'excepcio noInsercio");  assertEquals(2, th.mida(), "S'ha afegit 1 element i la mida no es correspon.");  assertDoesNotThrow(() -> th.inserir(c.getDni(), c), "El metode inserir() ha llençat l'excepcio noInsercio");  assertEquals(3, th.mida(), "S'ha afegit 1 element i la mida no es correspon.");  assertDoesNotThrow(() -> th.inserir(d.getDni(), d), "El metode inserir() ha llençat l'excepcio noInsercio");  assertEquals(4, th.mida(), "S'ha afegit 1 element i la mida no es correspon.");  assertDoesNotThrow(() -> th.inserir(e.getDni(), e), "El metode inserir() ha llençat l'excepcio noInsercio");  assertEquals(5, th.mida(), "S'ha afegit 1 element i la mida no es correspon.");  assertDoesNotThrow(() -> th.inserir(f.getDni(), f), "El metode inserir() ha llençat l'excepcio noInsercio");  assertEquals(6, th.mida(), "S'ha afegit 1 element i la mida no es correspon.");    boolean[] blcp = {false, false, false, false, false, false}; for(TADCiutada cp : th.obtenirValors()){ for(int i = 0; i < 6; i++){  if(!blcp[i] && lcp[i].compareTo(cp) == 0){ blcp[i] = true;  }  } }  boolean and = blcp[0]; for(int i = 1; i < 6; i++) { and = and && blcp[i];  }  assertTrue(and, "No s'han trobat tots els elements.");  /\* Canviar informació d'un element que ja existeix \*/  CiutadaPeu ncp = new CiutadaPeu("Informacio", "Canviada", "a"); assertDoesNotThrow(() -> th.inserir(ncp.getDni(), ncp), "Inserir ha llençat una excepció al afegir un element amb una clau repetida."); assertDoesNotThrow(() -> assertNotEquals(0,  th.obtenir("a").getNom().compareTo(a.getNom()), "L'element no te les seves dades canviades."),  "Inserir un element ha llençat la excepció noInsercio.");  /\* Comprovar que la taula es redimensiona si supera el limit (0.75) \*/    reiniciarTH();    int icapacity = th.getCapacity(); int imida = th.mida(); int i = 0;  while(icapacity == th.getCapacity()){  /\* (char)(i+103) = 'g' \*/ |

CiutadaPeu nncp = new CiutadaPeu(Integer.toString(i),

|  |
| --- |
| Integer.toString(i), (char)(i+103) + "");  assertDoesNotThrow(() -> th.inserir(nncp.getDni(), nncp), "No s'ha pogut inserir un ciutada."); i++;  }    assertEquals(i+imida, th.mida(), "La mida inicial mes els elements afegits no és igual a la mida actual.");  assertEquals(icapacity\*1.5, th.getCapacity(), "No s'ha redimensionat la taula hash correctament.");  assertTrue(th.obtenirFactorDeCarrega() <= 0.75, "El factor de carrega es superior al limit, s'hauria de redimensionar la taula.");  }    /\*\*   * Obtenir comprova si es troben tots els elements i s'hi s'obtenen correctament. * @throws noInsercio   \*/    @Test  void obtenir() throws noInsercio {  reiniciarTH(); /\* {a, b, c, d, e, f} \*/  assertDoesNotThrow(()->{ for(int i = 0; i < 6;i++){  TADCiutada tmp = th.obtenir(lcp[i].getDni()); assertNotNull(tmp, "L'element obtingut es null");  assertEquals(0, tmp.compareTo(lcp[i]), "L'element obtingut a la posició " + i + " no es l'esperat");  }  }, "No s'ha obtés un ciutada");    /\* Comprovar que salta una excepció si es busca un valor que no existeix \*/ assertThrows(noObtenir.class, () -> th.obtenir("Clau Inexistent"),  "Buscar una clau que no hi és a la llista no llença una excepció."); }    /\*\*   * Comprova que es puguin buscar tots els elemets i, s'hi l'element que s'està buscant forma part d'una colisió * també és troba. * @throws noInsercio   \*/ @Test  void buscar() throws noInsercio { reiniciarTH();  /\* Per com funciona el Hasher (XOR) si el codi es repeteix un nombre imparell de vegades el codi sera el mateix \*/  /\* hash("a") == hash("aaa") \*/  /\* Actualment la th te 6 ciutadans tots en posicions diferents \*/  /\* a->7, b->8, c->9, d->0, e->1, f->2 \*/  /\* Per tant, al ser "aaa" el mateix hash code que "a" s'afegira darrere  \*/  assertDoesNotThrow(() -> { for(int i = 0; i < 6; i++){  assertEquals(1, th.buscar(lcp[i].getDni()), "La disposicio dels |

elements no es la correcta.");

}

}, "No s'ha obtés un element que hi és a la taula.");

|  |
| --- |
| th.inserir("aaa", new CiutadaPeu("Nou", "Ciutada", "aaa"));    assertDoesNotThrow(() -> assertEquals(2, th.buscar("aaa"), "El nombre d'elements buscat no correspon amb l'esperat.")  , "No s'ha trobat un element colisionat");    assertThrows(noTrobat.class, () -> th.buscar("Clau Inexistent"), "No s'ha llençat una excepció per a un element inexistent.");    }    /\*\*   * Mida comprova que la mida augmenta i disminueix correctament. * @throws noInsercio   \*/ @Test  void mida() throws noInsercio { reiniciarTH();    assertEquals(6, th.mida(), "La mida quan es reiniciar la taula de hash ha de ser 6.");  th.inserir("aaa", new CiutadaPeu("Nou", "Ciutada", "aaa"));    assertEquals(7, th.mida(), "S'ha afegit 1 element i la mida no es correspon.");    th.crear();    assertEquals(0, th.mida(), "S'ha reiniciat la Taula i la mida no es correspon.");    }    /\*\*   * Es comprova que s'eliminin elements corectament en tots els casos possibles. * @throws noInsercio   \*/ @Test  void esborrar() throws noInsercio {  reiniciarTH(); /\* {a, b, c, d, e, f} \*/  assertDoesNotThrow(()->{ for(int i = 0; i < 6;i++){ reiniciarTH();  th.esborrar(lcp[i].getDni()); TADCiutada tmp = lcp[i];  assertThrows(noTrobat.class, ()->th.buscar(tmp.getDni()),  "L'element " + tmp + " no s'ha eliminat correctament."); }  }, "Esborrar ha llençat una excepció inesperada.");    reiniciarTH();  assertThrows(elementNoExisteix.class, ()->th.esborrar("Clau Inexistent"),  "No s'ha llençat una excepció al buscar un element inexistent.");  /\* Eliminar un element que colisiona amb un altre \*/ |

CiutadaPeu ncp = new CiutadaPeu("Nou", "Ciutada", "aaa");

|  |
| --- |
| th.inserir(ncp.getDni(), ncp);    /\* Eliminar el "head" de la colisio \*/    assertDoesNotThrow(() -> th.esborrar("a"), "No s'ha trobat l'element buscat.");  assertThrows(noObtenir.class, () -> th.obtenir("a"), "L'element no s'ha eliminat correctament.");  assertDoesNotThrow(() -> assertEquals(1, th.buscar("aaa"), "No s'ha trobat l'element"),  "No s'ha trobat el següent element de la colisió.");  /\* Eliminar el "tail" de la colisio \*/    reiniciarTH();  th.inserir(ncp.getDni(), ncp);  assertDoesNotThrow(() -> th.esborrar("aaa"), "No s'ha trobat l'element buscat.");  assertThrows(noObtenir.class, () -> th.obtenir("aaa"), "L'element no s'ha eliminat correctament.");  assertDoesNotThrow(() -> assertEquals(1, th.buscar("a"), "No s'ha trobat l'element"),  "No s'ha trobat el primer element de la colisió.");  /\* Eliminar un element en el mig d'una colisio \*/  CiutadaPeu ncp2 = new CiutadaPeu("Nounou", "Ciutadaciutada", "aaaaa"); reiniciarTH();  th.inserir(ncp.getDni(), ncp); th.inserir(ncp2.getDni(), ncp2); /\* Element del mig "aaa" \*/  assertDoesNotThrow(() -> th.esborrar("aaa"), "No s'ha trobat l'element buscat.");  assertThrows(noObtenir.class, () -> th.obtenir("aaa"), "L'element no s'ha eliminat correctament.");  assertDoesNotThrow(() -> assertEquals(1, th.buscar("a"), "No s'ha trobat l'element"),  "No s'ha trobat el primer element de la colisió."); assertDoesNotThrow(() -> assertEquals(2, th.buscar("aaaaa"), "No s'ha trobat l'element"),  "No s'ha trobat el següent element de la colisió.");  }    /\*\*   * Es comprova que esl valors obtinguts siguin tots els que hi ha present a la llista. * @throws noInsercio   \*/ @Test  void obtenirValors() throws noInsercio { reiniciarTH(); /\* {a, b, c, e, f} \*/ DLL<TADCiutada> dllcp = th.obtenirValors();  boolean[] blcp = {false, false, false, false, false, false}; for(TADCiutada cp : th.obtenirValors()){ for(int i = 0; i < 6; i++){  if(!blcp[i] && lcp[i].compareTo(cp) == 0){ |

blcp[i] = true;

}

}

|  |
| --- |
| }  boolean and = blcp[0]; for(int i = 1; i < 6; i++) { and = and && blcp[i];  }    assertTrue(and, "No s'han trobat tots els elements.");  /\* Obtenir valors en colisió \*/    CiutadaPeu ncp = new CiutadaPeu("Nou", "Ciutada", "aaa");  CiutadaPeu ncp2 = new CiutadaPeu("Nounou", "Ciutadanou", "aaaaa"); CiutadaPeu[] lcp2 = {a, b, c, d, e, f, ncp, ncp2};  th.inserir(ncp.getDni(), ncp); th.inserir(ncp2.getDni(), ncp2);    boolean[] blcp2 = {false, false, false, false, false, false, false, false};  for(TADCiutada cp : th.obtenirValors()){ for(int i = 0; i < 8; i++){  if(!blcp2[i] && lcp2[i].compareTo(cp) == 0){ blcp2[i] = true;  }  } }  and = blcp2[0];  for(int i = 1; i < 8; i++) { and = and && blcp2[i];  }    assertTrue(and, "No s'han trobat tots els elements."); }    /\*\*   * Es comprova que totes les claus obtingudes es corresponguin amb totes les claus que hi haurien d'haber. * @throws noInsercio   \*/ @Test  void obtenirClaus() throws noInsercio { reiniciarTH(); /\* {a, b, c, d, e, f} \*/    DLL<String> dllccp = th.obtenirClaus(); for(String s : dllccp){  assertDoesNotThrow(() -> th.buscar(s), "Les claus retornades no son correctes");  }    /\* Obtenir claus en colisió \*/    CiutadaPeu ncp = new CiutadaPeu("Nou", "Ciutada", "aaa");  CiutadaPeu ncp2 = new CiutadaPeu("Nounou", "Ciutadanou", "aaaaa");    th.inserir(ncp.getDni(), ncp); th.inserir(ncp.getDni(), ncp2);    dllccp = th.obtenirClaus(); |

for(String s : dllccp){

assertDoesNotThrow(() -> th.buscar(s), "Les claus retornades no son

|  |
| --- |
| correctes");  }    }    /\*\*   * Reinicia la taula de hash per tal de no repetir codi. * @throws noInsercio   \*/  private void reiniciarTH() throws noInsercio { th = new TaulaHash<>(); for(CiutadaPeu cp : lcp){ th.inserir(cp.getDni(), cp);  }  }  } |