Lista

```
public interface SimpleLinkedListADT<T> {

   public T removeFirst()
   public T removeLast()
   public T remove(T elem)
   public T first()
   public T last()
   public boolean contains(T elem)
   public boolean isEmpty()
   public int size()
   public Iterator<T> iterator()
}
```

Pila

```
public interface StackADT<T> {
      public void push(T elem)
      public T pop()
      public T peek()
      public boolean isEmpty()
      public int size()
}
```

Ilara / Cola

```
public interface QueueADT<T> {
      public void insert(T elem)
      public T remove()
      public T first()
      public boolean isEmpty()
      public int size()
}
```

Hash taula / Tabla hash

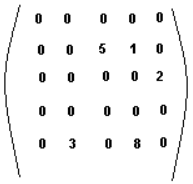
```
public interface Map<K, V> {
      public V put(K key, V value)
      public V get(K key)
      public T remove(K key)
      public boolean containsKey(K key)
      public int size()
}
```

DATU-EGITURAK ETA ALGORITMOAK

UZTAILA 2013

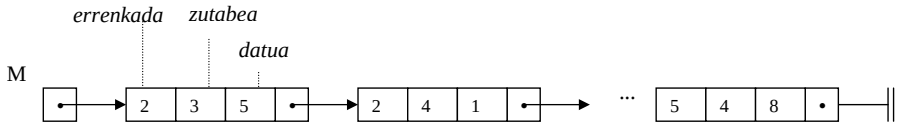
1. Matrize sakabanatua (2,5 puntu)

Matrize sakabanatuetan elementu gehienek zero balioa dute. Adibidez:

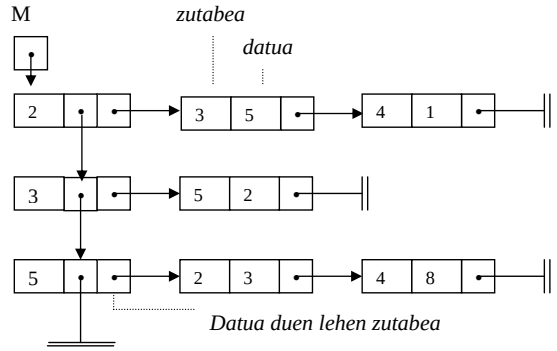


Mota honetako matrize bat era askotan adierazi daiteke:

- **A adierazpena**, zerrenda sinple bat, zero ez diren elementuekin, errenkada eta zutabez ordenatuta:



- **B adierazpena**, hutsak ez diren errenkaden zerrenda baten bidez. Errenkada bakoitzeko, zero ez diren zutabeak adieraziko dira, zutabearen zenbakia eta datua adieraziz. Errenkada bateko elementuak zutabearen arabera daude ordenatuta.



Hau eskatzen da:

a) A eta B adierazpenen klaseen definizioak idatzi.

b) Ondoko azpiprograma inplementatu:
public MatrizeB bihurtu(MatrizeA m)
matrizea sortuko du m matrizetik abiatuta, hau da, A adierazpenetik B adierazpenera bihurketa egingo du.

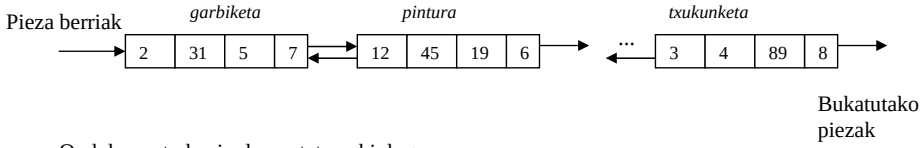
Algoritmo horrek O(N) kostukoa izan beharko du, N zero ez diren elementuen kopurua delarik.

2. Fabrika (1,5 puntu)

Fabrika batean autoen piezak egiteko kontrol-sistema bat ezarri nahi da. Fabrikak 30 prozesu-zentro ditu, eta piezak era honetan prozesatzen dira:

- Pieza berri bat prozesatzen hasteko, garbiketa zentroan sartuko da.
- Une horretatik aurrera, piezak zentro batetik bestera mugituko dira. Antolaketa-arrazoiengatik, piezak bakarrik mugituko dira zentro batetik bere ondoan dagoen beste zentro batera.
- Piezak bi norabideetan mugi daitezke, hau da, A eta b ondoz-ondoko zentroak izanda, piezak Atik Bra edo Btik Ara mugi daitezke.
- Bukatutako piezak txukunketa zentrotik aterako dira.

Ondoko irudiak adibide bat erakusten du:



Ondoko metodoa inplementatu nahi dugu:

```
public class Mugimendua {

    Integer zentroa; // balio posibleak: “srrera”, “garbiketa”, “pintura”,
                     //        “lisaketa”, ... “txukunketa”, “irteera”
                     // osokoekin kodetua (-1 -> “sarrera”,
                     // 1 -> “garbiketa”, 30 ->“ txukunketa”, 31 -> “irteera”)
    Integer num;
          // baldin zentroa = “sarrera” orduan sartzen den pieza bat da
          //                (garbiketa-zentrora)
          // baldin zentroa = “irteera” orduan pieza bat aterako da
          //        (txukunketa-zentrotik). Kasu honetan, “num”-en
          //        balioa ez dago definituta
          // bestela, “num”-ek zentro batetik bestera mugituko den
          //                pieza-kopurua adierazten du
    String norabidea; // balio posibleak: “ezk” edo “esk”,
                      // piezen mugimendua adierazten du
                      // atributu honek ez du zentzurik zentroa “sarrera”
                      //    edo “irteera” bada
}

public class Fabrika {

   public SimpleLinkedList<Integer>
                      prozesatu(SimpleLinkedList<Mugimendua> lista)
   // Aurrebaldintza: “lista”-k gertatutako ebentoak ditu
   // Postbaldintza: emaitza bukatutako piezen zerrenda izango da
   //               (“irteera” moduko ebentoetatik)
}
```

Adibidez, sarrerako zerrendak ebento hauek izan ditzake:

```
-1 47       // “47” pieza sartzen da (garbiketa-zentrora)
-1 23       // “23” pieza sartzen da (garbiketa-zentrora)
-1 35       // “35” pieza sartzen da (garbiketa-zentrora)
1 2 esk     // 2 pieza pasako dira garbiketa-zentrotik (0) bere eskuinekora
...
-1 147      // “147” pieza sartzen da (garbiketa-zentrora)
31          // pieza bat ateratzen da (txukunketa-zentrotik)
2 3 ezk     // 3 pieza pasako dira pintura-zentrotik (1) bere ezkerrekora
...
-1 88       // “88” pieza sartzen da (garbiketa-zentrora)
31          // pieza bat ateratzen da (txukunketa-zentrotik)
...
```

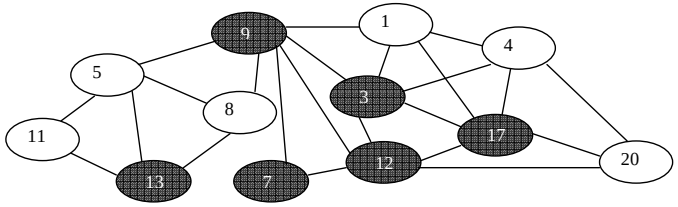
“prozesatu” metodoa inplementatu behar da. Horretarako, Bilara datu-mota abstraktua erabil daiteke..

Bilara:

```
public class Bilara<T> {


   public Bilara(); // eraikitzailea
   // bilara hasieratzen du (hutsa)

   public boolean hutsaDa();
   // true B hutsa baldin bada eta false bestela.

   public void txertatuEzk(T elemento);
   // E elementua ezkerretik gehitu dio

   public void txertatuEsk(T elemento);
   // E elementua eskuinetik gehitu dio

   public void ezabatuEzk();
   // ezkerrean dagoen elementua ezabatu du

   public void ezabatuEsk();
   // eskuinean dagoen elementua ezabatu du

   public T lortuEzk();
   // ezkerrean dagoen elementua bueltatzen du

   public T lortuEsk();
   // eskuinean dagoen elementua bueltatzen du

}
```

3. Jokoa (1,5 puntu)

Ondoko irudiak joko bat adierazten du.



Jokoaren helburua hasierako adabegi batetik bukaerako adabegi batera iristea badagoen asmatzea da. Horretarako jokoaren arau nagusia errespetatu behar da:

- Kolore bateko lauki batetik bakarrik beste koloreetako laukietara bakarrik pasa daiteke (hau da, lauki beltzetik zurira, edo zuritik beltzera).

Algoritmo bat egin nahi dugu, jokoan agertzen diren bi lauki emanda, bi elementu horiek konektatzen dituen bide minimoa emango diguna. Algoritmoak hau bueltatuko du:

- Zerrenda hutsa, laukien arteko konexiorik ez dagoenean
- Luzera minimoko bide bat baino gehiago balego, orduan edozein bueltatu daiteke

Adibidez, bilatuBidea("11", "20") deiak <11, 13, 8, 9, 1, 3, 4, 17, 20> zerrenda bueltatuko luke (egon daitezke luzera horretako edo gehiagoko beste bide batzuk).

```
public class Laukia {
    String kolorea; // "zuria" edo "beltza"
    int    balioa;
}

public class GraphAL<T> implements GraphADT<T>
{
   protected final int DEFAULT_CAPACITY = 100;
   protected int numVertices;   // number of vertices in the graph
   protected boolean[][] adjMatrix;   // adjacency matrix
   protected T[] vertices;   // values of vertices

}

public class GrafoJoko extends GraphAL<Laukia> {

   public SimpleLinkedList<Laukia> bilatuBidea(Laukia hasiera, Laukia bukaera)

}
```

4. Ebaluatzailea (1,5 puntu)

Era honetako aginduen sekuentzia dugu: "x = y + z". Hash-taula batean aldagaien balioak daude. Programa bat egin nahi dugu, aginduak ebaluatu eta hash-taula eguneratzeko.

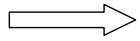
```
public class Agindua {
   String ize1;
   String ize2;
   String iz3;
   String eragile; // balio hauetatik bat: "+", "-", "/" eta "*"
   // era honetako agindua adierazten du:
   //                          ize1 = nombre2 ize ize3
}

public class AldagaienHashTaula extends HashMap<String, Integer> {

   public void ebaluatu(SimpleLinkedList<Agindua>)
}
```

Adibidez, aginduen zerrenda hau emanda:

Aginduen zerrenda: (x = x + j; contNum = contNum + x; y = contNum + x)

Hash-taula era honetan aldatuko litzateke:

0	j	5
1	x	3
2		
3	contNum	0
4	y	7
5		



0	j	5
1	x	8
2		
3	contNum	8
4	y	16
5		

Ondokoa eskatzen da:

- Ebaluatu metodoa inplementatu.
- Bere kostua kalkulatu **era arrazoituan**.