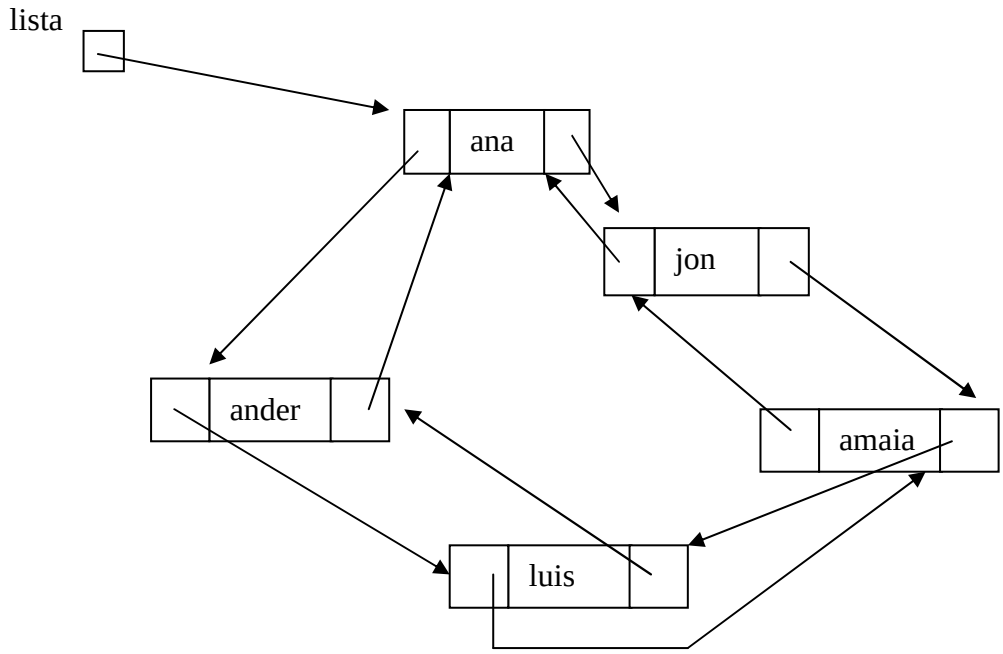
# ESTRUCTURAS DE DATOS Y ALGORITMOS

## ENERO 2013

**1. (1,5 puntos)** Tenemos una lista doblemente ligada circular:




Se quiere implementar la siguiente función:

```
public DoubleNode<T> {
   T data;
   DoubleNode<T> next;
   DoubleNode<T> prev;
}

public DoubleLinkedList<T> {

   DoubleNode<T> first;

   public T obtenerSuperviviente(Integer salto)
   // pre: la lista tiene al menos un elemento
   // post: devuelve un elemento, que será
   //      el resultado de ir eliminando de la lista cada vez el elemento
   //      correspondiente a "salto" (en el sentido de las agujas del reloj),
   //      hasta que quede un solo elemento

}
```

Por ejemplo, la llamada a obtenerSuperviviente(4) devolverá "jon" (el elemento que queda después de eliminar sucesivamente a ander, luis, ana y amaia).

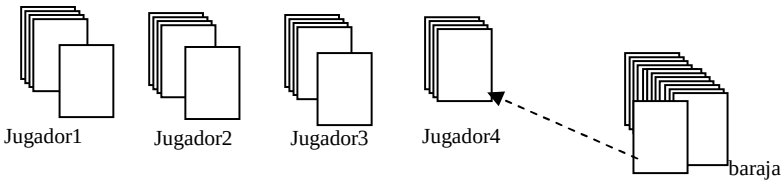
Se pide:

- Implementar el algoritmo
- Establecer de manera razonada el coste del algoritmo.

***2. Partida de cartas (1,5 puntos)***

Cuatro amigos van a jugar al juego de cartas de los SEISES. Las características del juego son las siguientes:

1. La baraja contiene 40 cartas divididas en cuatro palos (oros, copas, espadas y bastos). Cada palo consta de 10 cartas numeradas del 1 al 10.
2. Las cartas se reparten en 4 montones, es decir, 10 cartas por jugador.




3. El funcionamiento de la partida es el siguiente:
    - El primer jugador juega con la primera carta de su montón,
        - Si <u>puede la coloca</u> en la mesa.
        - Si no puede colocar la carta en la mesa, la deja de nuevo en el fondo de su montón
    - Y pasa el turno al siguiente jugador.

4. Un jugador <u>puede colocar</u> su carta en la mesa en 2 casos:
    - La carta es un 6.
    - La carta no es 6, pero es una carta consecutiva a otra que sí se encuentra en la mesa. Por ejemplo, si tengo el 2 de bastos y en la mesa está el 3 de bastos, ó si tengo el 8 de oros y en la mesa está el 7 de oros, etc.

- El juego acaba cuando un jugador se coloca la última carta de su montón, es decir, se queda sin cartas.

Utilizando los TADs Baraja y Bicola, **(no hay que implementar ninguna operación de los TAD's) se pide:**

**a)** Implementar las estructuras de datos necesarias para representar a los jugadores y la mesa con el estado del juego.

**b)** *Diseñar y escribir* un subprograma que simule una partida diciendo al final cuál ha sido el jugador ganador.

***Baraja***:

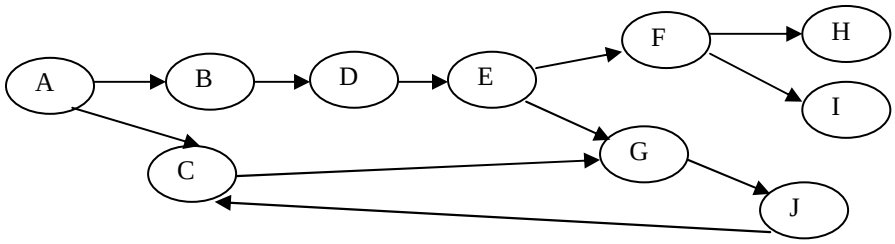
```
public class Carta {

   String palo;  // oros, copas, espadas, espadas
   int valor;    // valor entre 1 y 10
}

public class Baraja {

   prívate Carta[] cartas;

   public Baraja(); // constructora
   // postcondición: la baraja contiene 40 cartas en orden aleatorio

   public Iterator<Carta> iterador()
   // devuelve un iterador para recorrer las cartas

}
```

***Bicola***:

```
public class Bicola<T> {

   public Bicola(); // constructora
   // Inicializa la bicola (vacía)

   public boolean estaVacia();
   // Indicará si la bicola está o no vacía.

   public void insertarIzq(T elemento);
   // añade el elemento E por el extremo izquierdo de la bicola

   public void insertarDer(T elemento);
   // añade el elemento E por el extremo derecho de la bicola

   public void eliminarIzq();
   // borra el elemento del extremo izquierdo de la bicola

   public void eliminarDer();
   // borra el elemento del extremo derecho de la bicola

   public T obtenerIzq();
   // obtiene el elemento.del extremo izquierdo de la bicola

   public T obtenerDer();
   // obtiene el elemento.del extremo derecho de la bicola

}
```

***3. Callejero (1,5 puntos)***

Se tiene un grafo dirigido que representa las calles de una ciudad, que presentamos parcialmente en la siguiente figura:




Cada nodo representa una calle, y un arco entre dos nodos indica que desde la calle inicial se puede acceder a la calle destino.

Queremos desarrollar un algoritmo que, dadas dos calles, nos devuelva un camino de longitud mínima que conecta esas dos calles, teniendo en cuenta que algunas de esas calles están en obras, por lo que no se puede circular por ellas. El algoritmo devolverá:

- La lista vacía si no hay ningún camino que las une
- En caso de que haya varios caminos de longitud mínima, bastará con devolver cualquiera de ellos. Se debe tener en cuenta que el camino no podrá pasar por las calles en obras.

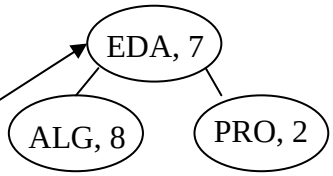
Por ejemplo, la llamada buscarCamino(“A”, “G”, <C, F, I>) devolverá la lista <A, B, D, E, G> (obsérvese que puede haber otros caminos de longitud igual o superior a este, y que el camino <A, C, G>, de longitud 3, no es una solución válida, ya que la calle C se encuentra en obras).

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

}

public class GrafoCallejero extends GraphAL<String> {
   public SimpleLinkedList<String> buscarCamino(String inicio, String fin,
                                                LinkedList<String> enObras)
}
```

***4. Calcular alumnos con mejor nota (1,5 puntos)***

Tenemos una tabla hash donde se guardan las notas obtenidas por los alumnos en distintas asignaturas. La clave de la tabla hash es el nombre y apellidos del alumno, y la información sobre las notas de las distintas asignaturas se guarda en un árbol binario de búsqueda, ordenado por el nombre de la asignatura.

Por ejemplo:




| | | |
|---|---|---|
| 0 | | |
| 1 | Jose Rodriguez | |
| 2 | | |
| 3 | Ana Agirre | |
| 4 | Nahia Perez | |
| 5 | | |

```
public class BinaryTreeNode<T> {
       protected T content;
       protected BinaryTreeNode<T> left;
       protected BinaryTreeNode<T> right;
}
public class BinarySearchTree<T> {
   protected int count;
   protected BinaryTreeNode<T> root;
}

public class Asignatura {
   int nota;
   String nombre;
}

public class ArbolAsignaturas extends BinarySearchTree<Asignatura> {
   // El árbol está ordenado alfabéticamente por el nombre de la asignatura

   public Integer mejorNota()
   // Post: el resultado es el valor de la mejor nota del árbol
}

public class ListaAlumnos extends HashMap<String, ArbolAsignaturas> {

   public SimpleLinkedList<String>
                        obtenerAlumnosConMejorNota(SimpleLinkedList<String> l)
   // Post: el resultado es la lista de alumnos de la lista l
   //       con la mejor nota
}
```

Se pide:

- Implementar el método `obtenerAlumnosConMejorNota`, que dada una lista de alumnos devuelve la lista con los que han obtenido la mejor nota (en cualquier asignatura). Por ejemplo, si la mayor nota obtenida es un 8, el algoritmo devolverá una lista con las personas de la lista que han sacado un 8.
- Establecer de manera razonada el coste del algoritmo.

***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()
}
```

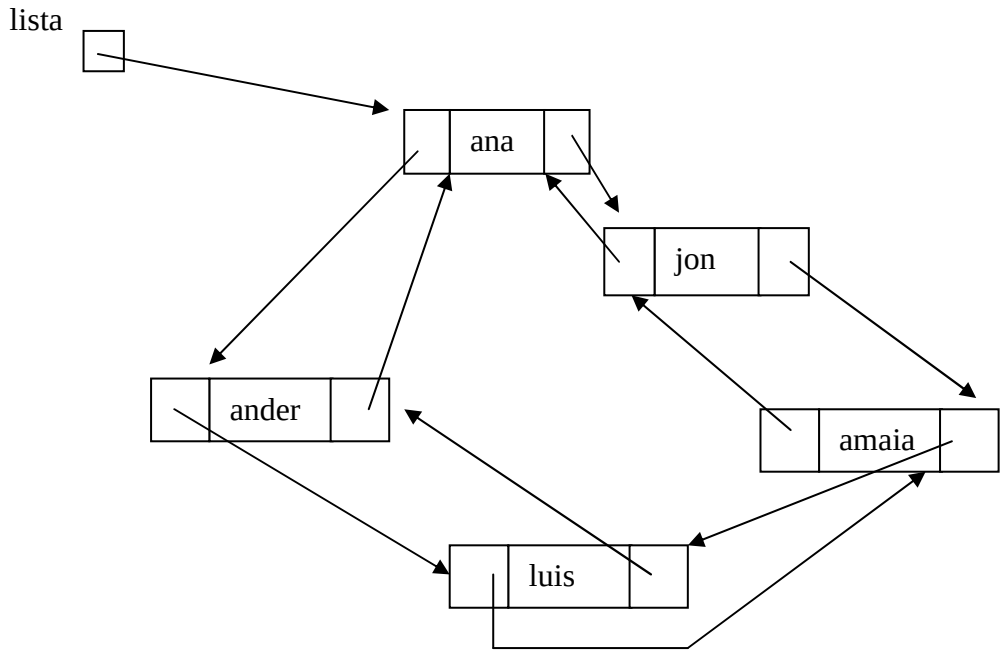
***Hash taula / Tabla hash***

```
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

## URTARRILA 2013

**2. (1,5 puntu)** Estekadura bikoitzeko lista zirkularra dugu:




Funtzio hau inplementatu nahi dugu:

```
public DoubleNode<T> {
   T data;
   DoubleNode<T> next;
   DoubleNode<T> prev;
}

public DoubleLinkedList<T> {

   DoubleNode<T> first;

   public T azkenaLortu(Integer jauzi)
   // aurre: listak gutxienez elementu bat du
   // post: elementu bat bueltatuko du, zerrendatik pausu bakoitzean
   //       elementu bat kenduz, unekotik "jauzi" elementu pasatuz
   //       (erlojuko orratzen norabidean) elementu bakarra geratu arte

}
```

Adibidez, azkenaLortu(4) deiak "jon" bueltatuko du (elementu hauek ezabatu ondoren: ander, luis, ana eta amaia).

Hau eskatzen da:

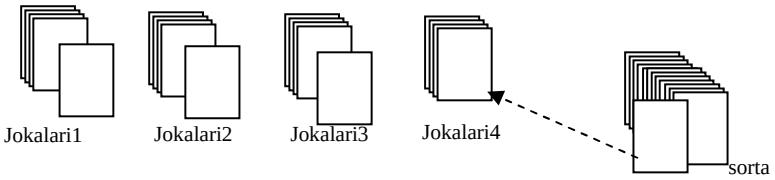
- Algoritmoa inplementatu
- Kostua kalkulatu, modu arrazoituan.

***2. Karta jokaldia (2,5 puntu)***

Lau lagunek SEIKOEN jokaldia egin nahi dute. Hauek dira jokoaren ezaugarri nagusiak:

1. Karta sortak 40 karta ditu, lau sailetan banatuta (urreak, kopak, ezpatak eta bastoiak). Sail bakoitzak 10 karta ditu, 1etik 10era.

2. Kartak lau jokalarien artean banatuko dira, hau da, jokalari bakoitzak 10 karta izango ditu.




3. Partida honela antolatuko da:
   - Lehen jokalariak bere lehen karta hartuko du (gainekoa),
     - Ahal badu (ikus behean), mahaian jarriko du.
     - Ezin badu, orduan karta hori bere karta guztien azpian jarriko du
   - Eta hurrengo jokalariari pasako zaio txanda.

4. Jokalari batek ondoko kasuetan jar dezake karta mahaian:
   - Karta seiko bat da.
   - Karta ez da seikoa, baina mahaian badagoen karta baten ondorengoa da. Adibidez, nire karta urrezko bikoa baldin bada, eta mahaian urrezko hirukoa balego, edo bestela nire karta urrezko zortzikoa balitz eta mahaian urrezko zazpikoa balego.

- Jokoa jokalari batek bere azken karta jartzen duenean amaituko da

Karta_Sorta eta Bilara DMAak erabilita (ez dira inplementatu behar DMA hauen eragiketak) ondokoa eskatzen da:

**a)** Problema hau ebazteko erabiliko dituzun datu-egiturak definitu (jokalariak eta mahaiaren egoera adierazteko).

**b)** ***Diseinatu eta idatzi*** azpiprograma bat partida bat simulatu eta irabazlea zein den esango diguna.

KartaSorta DMA:

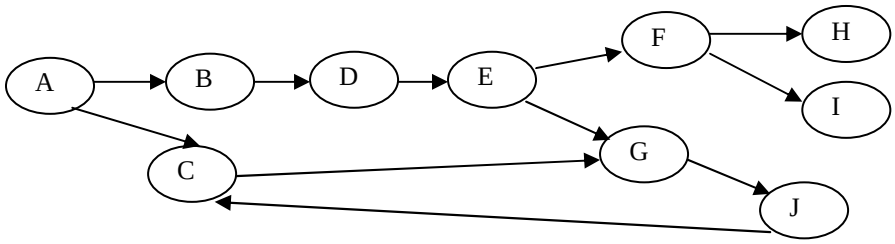
```
public class Karta {

   String palo;  // urrea, kopa, ezpata, bastoia
   int balioa;    // 1 eta 10 arteko balioa
}

public class KartaSorta {

   prívate Karta[] kartak;

   public KartaSorta (); // eraikitzailea
   // post: 40 karta daude ausaz

   public Iterator<Carta> iteradore()
   // kartak aztertzeko iteradorea bueltatzen du

}
```

***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. Kaleak (1,5 puntu)***

Grafo zuzendu batek hiri bateko kaleak adierazten ditu, ondoko irudian ikus daitekeen bezala:




Adabegi bakoitzak kale bat adierazten du, eta bi adabegiren arteko arku batek lehen kaletik bigarrenera joan daitekeela adierazten du.

Algoritmo bat garatu nahi dugu esateko, bi kale emanda, zein den kale horiek konektatzen dituen luzera minimoko bide bat, kale batzuk obratan daudela kontuan hartuta, eta ezin dela kale horietatik pasa. Algoritmoak hau bueltatuko du:

- Zerrenda hutsa, bi kale horiek ezin baldin badira lotu
- Luzera minimoko bide bat baino gehiago balego, edozein bueltatuko da. Kontuan izan behar da bidea ezin dela pasa obratan dauden kaleetatik.

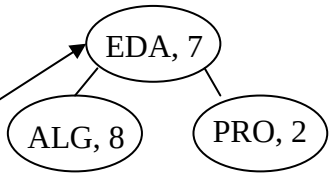
Adibidez, bilatuBidea(“A”, “G”, <C, F, I>) deiak <A, B, D, E, G> zerrenda bueltatuko du (kontuan izan <A, C, G> bidea ez dela soluzio bat, C kalea obratan dagoelako).

```
public class GraphAL<T> implements GraphADT<T>
{
   protected final int DEFAULT_CAPACITY = 100;
   protected int             numVertices;   // number of vertices in the graph
   protected LinkedList<Integer>[] adjList;  // adjacency list
   protected T[]                   vertices; // values of vertices

}

public class GrafoKaleak extends GraphAL<String> {
   public SimpleLinkedList<String> bilatuBidea(String hasiera, String bukaera,
                                               LinkedList<String> obratan)
}
```

***4. Lortu nota hobereneko ikasleak (1,5 puntos)***

Hash taula bat dugu ikasleek lortutako notak gordetzeko. Gakoa ikaslearen izen-abizenak dira, eta irakasgaien noten informazioa bilaketa-zuhaitz bitar batean gordetzen da, irakasgaiaren izenaren arabera ordenatuta.

Adibidez:




| | | |
|---|---|---|
| 0 | | |
| 1 | Jose Rodriguez | |
| 2 | | |
| 3 | Ana Agirre | |
| 4 | Nahia Perez | |
| 5 | | |

```
public class BinaryTreeNode<T> {
       protected T content;
       protected BinaryTreeNode<T> left;
       protected BinaryTreeNode<T> right;
}
public class BinarySearchTree<T> {
   protected int count;
   protected BinaryTreeNode<T> root;
}

public class Irakasgaia {
   int nota;
   String izena;
}

public class IrakasgaienZuhaitza extends BinarySearchTree<Irakasgaia> {
   // Zuhaitza irakasgaiaren izenaren arabera alfabetikoki ordenatuta dago

   public Integer notaHoberena()
   // Post: emaitza zuhaitzeko nota hoberena da
}

public class Ikasleak extends HashMap<String, IrakasgaienZuhaitza> {

   public SimpleLinkedList<String>
                       lortuNotaHoberenekoIkasleak(SimpleLinkedList<String> l)
   // Post: emaitza l zerrendako nota hoberena duten ikasleen izenak dira
}
```

Ondokoa eskatzen da:

- `lortuNotaHoberenekoIkasleak` metodoa inplementatu, ikasleen zerrenda bat emanda, nota hoberena lortu duten ikasleen zerrenda bueltatuko duena (edozein irakasgaitan). Adibidez, demagun 8 dela nota maximoa irakasgai guztietan, eta kasu horretan 8 atera dutenen zerrenda aterako litzateke.
- Kostua kalkulatu, modu arrazoituan.