(J94) Al comenzar una partida de póker cada uno de los tres jugadores **A**, **B** y **C** poseía un montón de 100 billetes de 1 dólar sobre la mesa, pero ... ¡ojo! el jugador **A** aparte de un excelente tahur también es un impecable falsificador. ¡Todos sus billetes eran falsos!

Sabiendo que los pagos realizados durante la partida fueron recogidos en el fichero de texto F_PAGOS, **desarrollar un programa que determine con cuántos billetes legales se marcha el falsificador (jugador A) al finalizar la partida.**

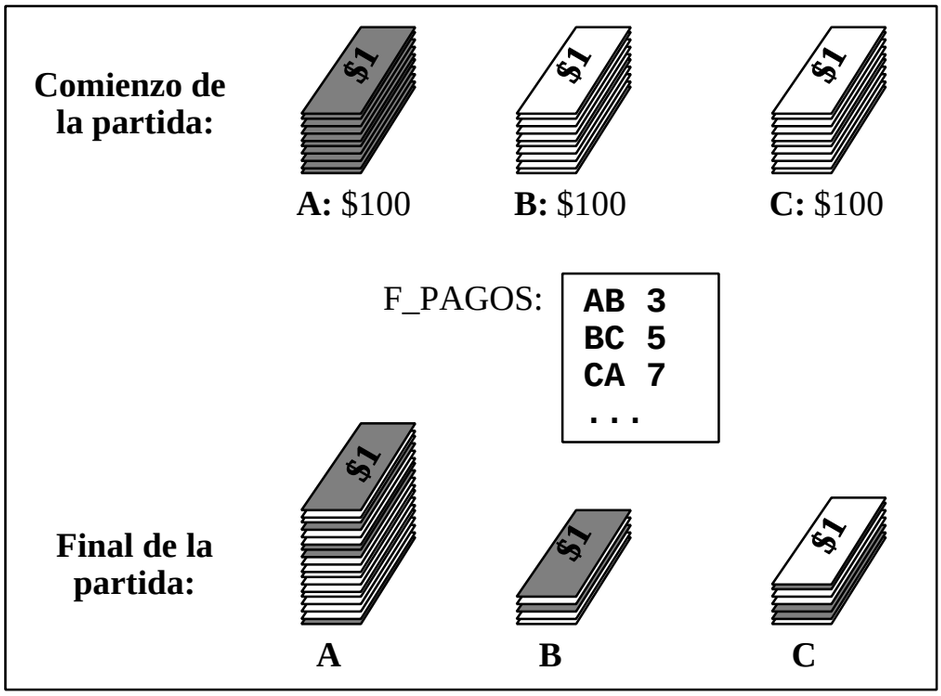
En el fichero F_PAGOS cada pago se representa en una línea con el siguiente formato: el primer carácter representa al jugador pagador, el segundo al jugador que cobra y la cantidad pagada viene a continuación separada por blancos. Por ejemplo:

```
AB  5
```

El jugador **A** paga al jugador **B** la cantidad de 5 dólares.

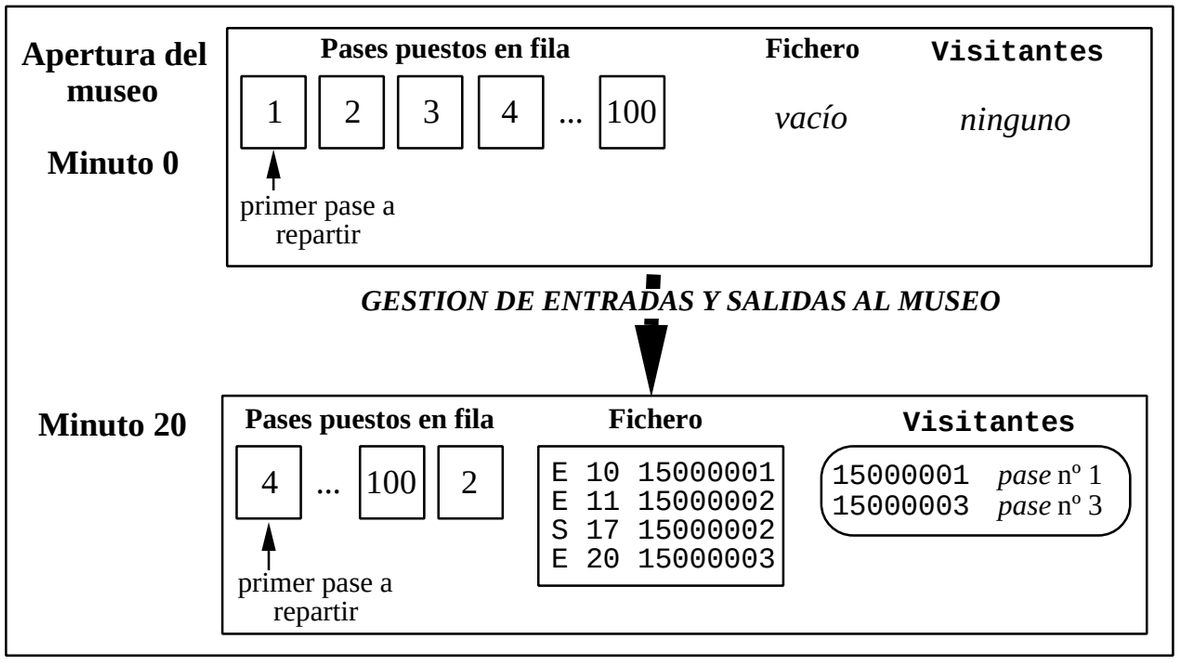
Los pagos se realizan de 1 dólar en 1 dólar cogiendo el billete de encima del montón del jugador pagador y depositándolo encima del montón del jugador que cobra.

Utiliza alguno de los TADs que conoces.



(S94) Para entrar al Museo de las joyas de la Castafiore es preciso tener un pase numerado que se recoge y entrega en la portería. El portero que gestiona los pases va creando un fichero de texto con los datos de entrada y salida de visitantes. La gestión de los pases consiste en lo siguiente:

- Todas las mañanas antes de abrir (minuto cero) se colocan todos los pases en fila y en orden creciente del 1 al 100.
- Cuando entra una persona se le entrega *el primer pase de la fila* y se añade al fichero el movimiento de entrada correspondiente. Por ejemplo el movimiento: **`E 330 15999888`** significa que la persona de DNI **`15999888`** entró ('**`E`**') al museo en el minuto **`330`** (integer) tras la apertura del museo (minuto 0).
- Cuando sale una persona se le recoge el pase, se coloca éste *al final de la fila de pases* y se escribe el movimiento de salida correspondiente. Por ejemplo: **`S 360 15999888`** significa que la persona de DNI **`15999888`** salió ('**`S`**') en el minuto **`360`**.



Una noche, ante la sospecha de un robo y una vez que el fichero F_MVTOS ha sido cerrado con todos los movimientos del día, el detective del museo quiere saber qué personas estaban visitando el museo al término de un minuto concreto y cuál era el número del pase que portaba cada una (esto último, aunque misterioso para nosotros, resulta vital para sus investigaciones, insiste el detective).

Se pide:

a) **Escoger uno de los TAD conocidos** para representar la *fila de pases*.
Explicar por qué se elige y de qué tipo son sus elementos.

b) Nos dan la especificación de la clase Visitantes:

Clase Visitantes

```
public class Visitantes extends Object {
```

public void apertura ()
Inicializa el *grupo de visitantes* a vacío

public void entrada(Long dni, Integer pase)
Añade el visitante de *dni* y numero de *pase* dados al *grupo de visitantes*.

public void salida(Long dni)
Elimina del *grupo de visitantes* a aquél que tiene el *dni* dado.

ver_primero(in t_visitantes; out t_dni; out t_pase)
Da como salida el número de *dni* y el número de *pase* del visitante que fue introducido el primero respecto a los que están en este momento en el *grupo de visitantes* .

public Integer ver_pase(Long dni)
Dado un *dni* de un visitante que se encuentra en el *grupo de visitantes* devuelve el número de pase que le fue asignado.

public boolean hay_alguien()
Devuelve falso si el *grupo de visitantes* es vacío y cierto si hay algún visitante en ese grupo.

Utilizando el TAD elegido en el apartado a) y la clase Visitantes, parametriza, diseña e implementa un método de la clase Visitantes tal que dado un minuto y partiendo de la información del fichero F_MVTOS escriba en la salida estandar el DNI y el número de pase de las personas que se encontraban visitando el museo en el minuto dado.

(J95) El centro de cálculo de *MUTSOBITSI COMPUTERS* dispone de las impresoras **lps1**, **lps2**, **lps3**, **lps4** y **lps5** para la impresión de trabajos en *condiciones normales.*

En caso de *apagón* generalizado esas impresoras quedan desconectadas, pero el sistema de seguridad activa la impresora de emergencia **lps6** (que dispone de sistema eléctrico de alimentación autógeno) y redirige a ella todos los trabajos de impresión que habían quedado pendientes en las cinco impresoras.

El procedimiento para el redireccionamiento en caso de apagón consiste en volcar sucesivamente **todos** los trabajos de cada una de las impresoras, empezando por **lps1** y terminando por **lps5**, sobre la impresora **lps6**.

Mientras no se subsana el apagón, las *peticiones de trabajos* de impresión que vengan dirigidas a las impresoras **lps1**, **lps2**, **lps3**, **lps4** y **lps5** habrá que redirigirlas a la impresora de emergencia **lps6**.

Cuando se *subsana* el apagón, las impresoras normales se reactivan para poder recibir trabajos de impresión. Sin embargo, los trabajos que fueron redireccionados a **lps6** durante el apagón permanecerán allí hasta ser impresos.

Se pide:

1) **Escoger uno de los TADs conocidos** para representar el conjunto de trabajos a imprimir en una impresora. Indica de qué tipo son los elementos del TAD. **Justifica tu elección**.
2) Utilizando el TAD elegido, **diseña e implementa un programa** tal que a partir de los datos contenidos en el fichero **F_IMP.dat** muestre en el terminal cuál es la situación final de trabajos pendientes en cada impresora.

El fichero **F_IMP.dat** que nos facilitan, contiene los eventos de impresión acaecidos desde la puesta en marcha del sistema, en orden cronológico. **Ejemplo:**

F_IMP.dat

```
(1)→ Plps2 1
     Plps3 2
     Plps5 3
     Plps5 4
(2)→ Flps5
     Plps2 5
(3)→ A
     Flps6
     Plps1 6
(4)→ S
     Plps1 7
```

El evento de impresión **(1)** del fichero **F_IMP.dat** es de tipo *petición de impresión* (**P**), y la impresora a la que se dirige la solicitud es **lps2** El número **1,** que viene a continuación, es el número del trabajo de impresión.

El evento **(2)** corresponde a la *finalización*(**F**) del trabajo que se estaba imprimiendo en la impresora **lps5**

El evento **(3)** indica un *apagón* (**A**) en el sistema y el **(4)** que se ha *subsanado* (**S**) el apagón .

En el ejemplo anterior, tras tratarse el evento marcado con **(2)**, el estado de las peticiones en las impresoras es el siguiente:

lps1	**lps2**	**lps3**	**lps4**	**lps5**	**lps6**
- 1	2	-	4	-	

El estado anterior a tratar el evento **(3)**, de apagón, sería el que se muestra a continuación:

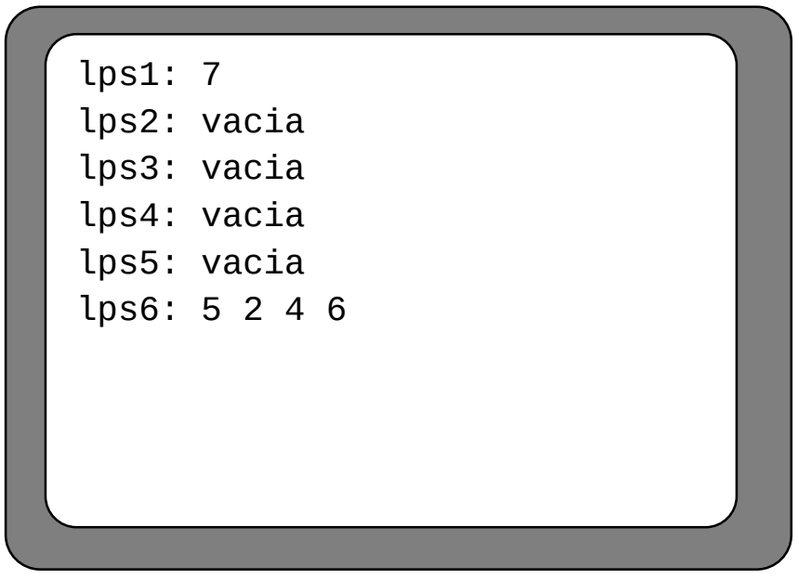
lps1	**lps2**	**lps3**	**lps4**	**lps5**	**lps6**
- 1	2	-	4	-	
5					

Tras tratar el evento **(3)** de apagón, la situación de los trabajos quedaría así [1]:

lps1	**lps2**	**lps3**	**lps4**	**lps5**	**lps6**
- -	-	-	-	1	(de lps2)
				5	(de ")
				2	(de lps3)
				4	(de lps5)

No habrá que considerar la posibilidad de errores en los datos contenidos en el fichero **IMP.dat** (por ejemplo casos como 2 apagones sin evento S entre ellos, ordenes de *finalización* a impresoras lps1 … lps5 durante un apagón, etc.)

La salida que debería escribir el programa pedido sobre el terminal, en el caso del ejemplo anterior, sería la siguiente:

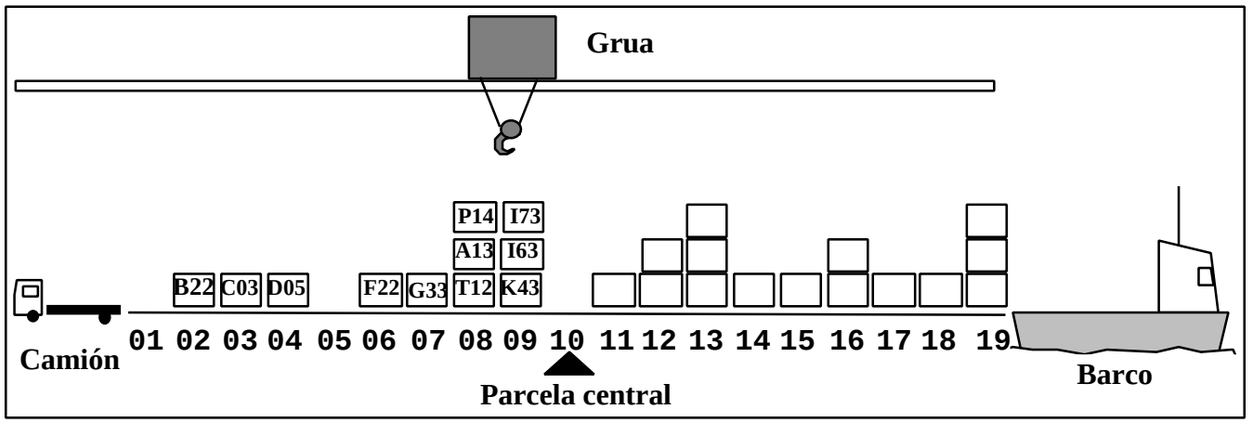


[1] Aquí hay que tener en cuenta el procedimiento para redireccionamiento en caso de apagón introducido anteriormente.

(S95) El puerto de Pasajes dispone de una *grúa* para descargar contenedores de un *camión*, almacenarlos temporalmente en el muelle y cargarlos en un *barco*.

La grúa es capaz de mover los contenedores de uno en uno entre cualesquiera de las **19** parcelas destinadas a su almacenaje. También puede coger contenedores de la posición donde se coloca el camión y dejarlos en la posición donde se coloca el barco.

La *parcela central* no se utiliza para almacenar contenedores de forma permanente. Su uso se destina a dejar momentáneamente contenedores cuando hay que desplazarlos de alguna parcela para sacar otro que está por debajo. Una vez se ha conseguido sacar el contenedor, todos los que han sido situados momentáneamente en la *parcela central* se vuelven a colocar en su parcela original.



Se pide:

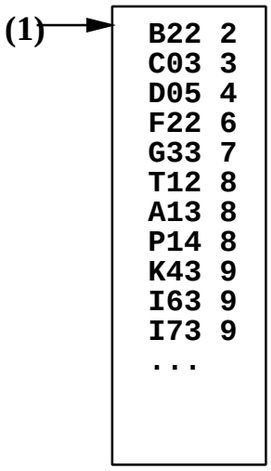
a) Seleccionar un TAD conocido para representar el conjunto de contenedores almacenados sobre una parcela. Justificar brevemente la selección.
b) Utilizando el TAD seleccionado diseñar e implementar un programa tal que dado el fichero de texto **F_GRUA.DAT** que posee una relación de cargas y descargas efectuadas, cree un fichero **F_GRUA.RES** que recoja cuáles han sido todos los traslados de contenedor efectuados por la grúa al ejecutar ese trabajo. La situación inicial de todas las parcelas se obtendrá utilizando el fichero de texto **F_INICIAL.DAT**

Suponer que las peticiones de carga y descarga que aparecen en el fichero **F_GRUA.DAT** siempre se pueden llevar a cabo.

Ejemplo:

El fichero utilizado para obtener la situación inicial de la figura es el siguiente:

F_INICIAL.DAT

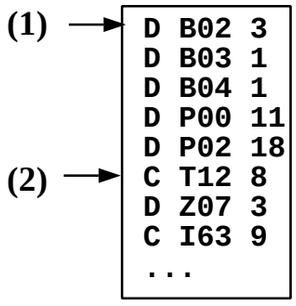


La línea **(1)** del fichero **F_INICIAL.DAT** representa que el contenedor **B22** en la situación inicial está situado en la parcela número **2**.

Los contenedores de cada parcela aparecen en el mismo orden en que fueron descargados de camiones. Esto supone que la primera línea del fichero correspondiente a una parcela concreta contiene el contenedor situado sobre la parcela, y la última línea correspondiente a esa parcela es la del contenedor de la parte superior.

Suponiendo que el fichero de cargas y descargas es el siguiente:

F_GRUA.DAT



La línea **(1)** del fichero **F_GRUA.DAT** representa la **descarga** (**D**) del contenedor **B02** en la parcela **3**. La descarga siempre se efectúa tomando el contenedor desde un *camión.*

La línea **(2)** corresponde a la carga (**C**) en el *barco* del contenedor **T12** que se encuentra en la parcela **8**.

El fichero resultado sería el siguiente:

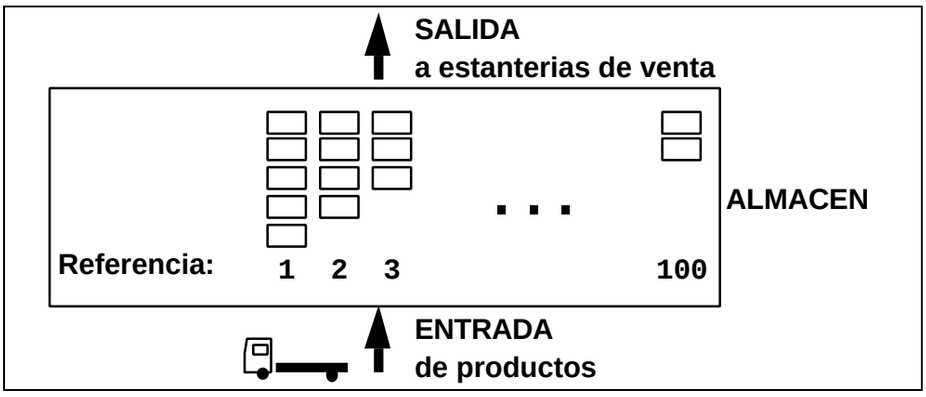
F_GRUA.RES

Mover contenedor de	Camión	a	3
Mover contenedor de	Camión	a	1
Mover contenedor de	Camión	a	1
Mover contenedor de	Camión	a	11
Mover contenedor de	Camión	a	18
Mover contenedor de	8	a	10
Mover contenedor de	8	a	10
Mover contenedor de	8	a	Barco
Mover contenedor de	10	a	8
Mover contenedor de	10	a	8
Mover contenedor de	Camión	a	3
Mover contenedor de	9	a	10
Mover contenedor de	9	a	Barco
Mover contenedor de	10	a	9
...			

(J96) El hipermercado *SALDOSKI* ha automatizado la gestión de su almacén donde se guardan los productos que tienen como referencia números del 1 al 100.

Todas las entradas y salidas de cajas de productos se van grabando de forma automática en el fichero **F_MOVIMIENTOS.DAT**. Por cada caja de producto que entra o sale del almacén se escriben su *referencia* y *fecha de caducidad* .

El almacén está organizado de forma que se saquen a estanterías para su venta aquellos productos cuya fecha de caducidad sea la más próxima al día en que se sacan y sólo se reciban productos con fecha de caducidad posterior a la de la última remesa.



Se pide:

a) Seleccionar un TAD conocido (y el tipo de los elementos que contiene) para representar las cajas de un producto existentes en el almacén. Justificarlo brevemente.

b) Utilizando el TAD seleccionado, **diseñar** e **implementar** un programa tal que dado el fichero de texto **F_INICIAL.DAT** que representa la situación inicial de existencias en el almacén y el fichero de texto **F_MOVIMIENTOS.DAT** que posee la relación de entradas y salidas efectuadas durante un día, cree los siguientes ficheros de texto:

i) **F_OFERTAS.DAT**: contiene información de las cajas a ser *sacadas* a estanterías al final de la jornada porque caducarán en un plazo inferior a 3 días.

ii) **F_PEDIDOS.DAT:** contiene un pedido de *reposición* por cada producto que cuente con menos de 5 cajas en el almacén.

Se supondrá que tanto las entradas como salidas de **F_MOVIMIENTOS.DAT** se pueden llevar siempre a cabo.

(S96) Todas las mañanas, en la oficina de Correos de Villa García, se hace un reparto de la correspondencia entre los diez carteros que forman parte de la plantilla. Cada uno de ellos tiene asociado un distrito postal (desde el **10.001** hasta el **10.010**).

El reparto de las cartas entre los carteros se hace de la siguiente manera:

- La asignación de cartas de hace guardando el orden de llegada a la oficina de Correos.
- Cada cartero podrá tener asignadas como máximo **200** cartas.
- Todas las mañanas comienza el reparto de cartas a los diferentes carteros según su distrito. Es decir, si una carta tiene distrito 10.004 se le asigna al cartero que tiene asociado dicho distrito. Esto es así, siempre y cuando al cartero en cuestión se le hayan asignado menos de 200 cartas.
- Las cartas que no se pueden asignar en el proceso descrito anteriormente se distribuirán entre los carteros con arreglo a otro tipo de criterios.

La oficina de Correos quiere que desarrollemos un programa que realice el reparto descrito anteriormente y obtenga información útil para realizar el reparto de las cartas que en su caso queden por asignar.

Se pide:

a) Seleccionar la o las **estructuras de datos** que permitan el reparto de la correspondencia y la obtención de las salidas que nos piden a continuación.
Tened en cuenta que se pueden usar los TADs conocidos.

b) Utilizando la estructura seleccionada, **diseñar** e **implementar** un programa tal que dado el fichero de texto **CARTAS.TXT** que contiene la información sobre las cartas a repartir, introducidas **por orden de llegada,** cree los siguientes ficheros de texto, donde la información aparece **agrupada por distrito** y entre las direcciones de un mismo distrito se **mantiene el orden de llegada**:

- **REPARTO.TXT** que contiene la información sobre las cartas que han sido asignadas a cada cartero.
- **RESTO.TXT** que contiene la información sobre las cartas que **no** han sido asignadas todavía.

En la cabecera de cada distrito aparece el **nº de cartas** que quedan por asignar de ese distrito (antes de escribir las direcciones). Si todas las cartas se han podido asignar ya al cartero, en su lugar figura el **nº de cartas** que se le podrían asignar a este hasta completar el total de 200 cartas asignables.

Formato de los ficheros

El formato del fichero de entrada **CARTAS.TXT** (direcciones en orden de llegada) es:

```
10002 Gran vía 3
10002 Bailén 20
10001 Sicilia 11
10002 Gran vía 3
  ...
```

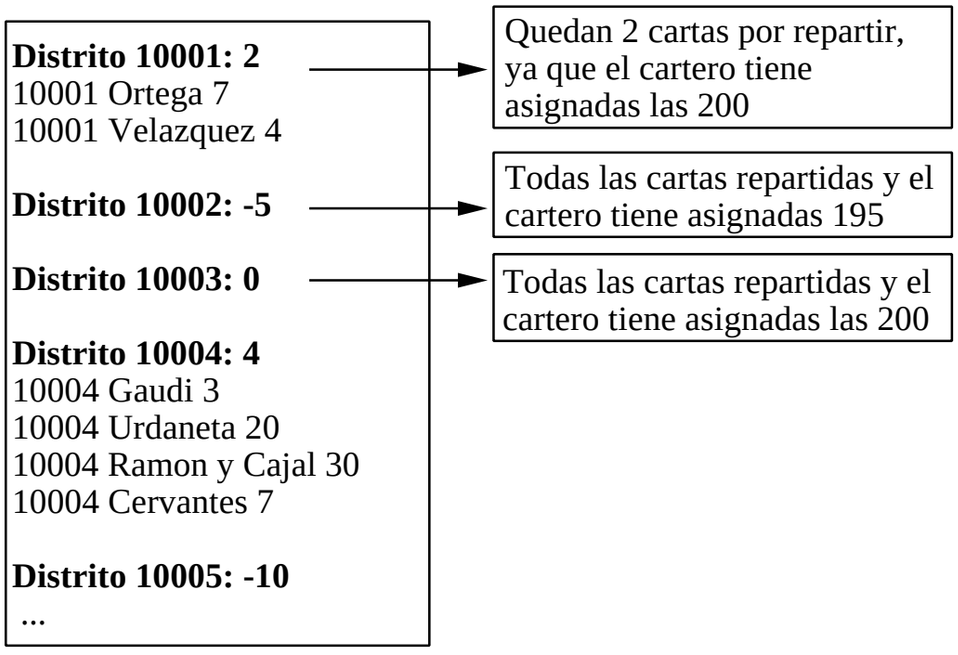
El formato del fichero de salida **REPARTO.TXT** (direcciones agrupadas por distrito. Las direcciones de cada grupo mantienen entre sì el orden de llegada a la oficina) es:

```
Cartero 10001
10001   Sicilia 11
10001   ...

Cartero 10002
10002   Gran vía 3
10002   Bailén 20
10002   ...

Cartero 10003
10003   ...
```

El formato del fichero de salida **RESTO.TXT** (direcciones agrupadas por distrito. Las direcciones de cada grupo mantienen entre sí el orden de llegada a la oficina. Las cabeceras incluyen el número de direcciones que se escriben por detrás, o en su caso el número de cartas que se podrían asignar al cartero hasta completar las 200) es:



(J97) En una vieja mansión escocesa vive una familia de fantasmas (cada uno de ellos tiene su DNI). Los duros tiempos de crisis han hecho que los dueños de la morada hayan decidido convertirla en hotel, disponiendo de **25** habitaciones. Ante la perspectiva de propinar sustos a los confiados huéspedes, los fantasmas se han organizado por habitaciones, de forma que:

- Cada uno de los fantasmas tiene asignada una sola habitación.
- Cada habitación puede tener asignados varios fantasmas diferentes.
- Una habitación recibe exactamente un susto por noche siempre y cuando haya algún fantasma asignado a ella.

Los fantasmas trabajan por riguroso orden: cuando una habitación está ocupada una noche, un fantasma asociado a esa habitación da un susto, aguardando para dar otro a que el resto de sus compañeros lo hayan hecho.

Todos los fantasmas tienen el mismo cupo de sustos: **10**. Cuando alguno consigue dar los 10 sustos, pasa a mejor vida y deja el hotel.

La oficina del hotel, en el fichero de texto **INICIO.TXT**, tiene información de la distribución inicial de los fantasmas en las habitaciones, junto con el número de sustos que lleva ya dados cada uno de ellos. Los primeros fantasmas que aparecen son los que antes pueden dar un susto. Además disponen de otro fichero de texto denominado **RESERVAS.TXT**, donde aparecen todas las reservas de las habitaciones durante una temporada.

Se pide:

a) Selecciona la o las **estructuras de datos** necesarias para la obtención de la salida. Ten en cuenta que puedes utilizar los TAD conocidos. Por cada TAD utilizado indica el tipo de sus elementos.

b) Utilizando la o las estructuras seleccionadas, desarrolla el programa que nos piden tomando los datos de los ficheros de texto **INICIO.TXT** y **RESERVAS.TXT**, y devolviendo por la **salida estándar** (pantalla):

- La relación de los DNI de los fantasmas que han pasado a mejor vida en esa temporada.
- El número de la habitación donde más fantasmas han pasado a mejor vida en esa temporada (si hubiera varias, una cualquiera de ellas).

Formato de los ficheros:

El fichero **INICIO.TXT** contiene una lÌnea por cada fantasma, en ***orden de prioridad para dar un susto***. La información de cada línea consta del DNI del fantasma, el número de sustos dado hasta el momento por el mismo y el número de habitación que tiene asignada.

```
12554231 2    14
13455904 1    25
15667231 8    1
...
19563211 9    14
```

DNI del fantasma — nº de sustos dados — nº de habitación

El fichero **RESERVAS.TXT** contiene una línea por reserva de habitación en ***orden de llegada de los huéspedes***. Cada línea consta del número de habitación y el número de noches de la reserva.

```
25    3
12    5
10    1
...
5     4
```

nº de habitación — nº de noches reservadas

Por la **salida estándar** se escribirá una línea por fantasma que pase a mejor vida (10 sustos en su haber), y la habitación que más fantasmas han pasado a mejor vida en esa temporada:

```
Fantasmas que han pasado a mejor vida:
13455904
19563211
...
habitación con más fantasmas hacia mejor vida:
14
```

(S97) Un centro instala una nueva centralita telefónica que le permite detectar cada número de teléfono al que se hace una llamada, la extensión desde la que se produce, además de medir el tiempo de cada llamada.

Nos piden que desarrollemos un programa para conocer cuál es **el teléfono al que más tiempo se llama** considerando el tiempo total de todas las llamadas realizadas al mismo. también interesa **conocer la información de todas las llamadas realizadas a ese teléfono**, esto es la extensión desde la que se llamaba y la duración de cada conversación, en el ***orden en el que se produjeron***.

Se pide:

a) Selecciona la o las **estructuras de datos** necesarias para la obtención de la salida ***si los teléfonos fueran un máximo de 100*** y además fuesen ***consecutivos*** (ejemplo desde el teléfono 943000000 al 943000100).
Trata de utilizar alguno de los TAD conocidos. Por cada TAD utilizado indica el tipo de sus elementos.

b) Utilizando la o las estructuras seleccionadas en el apartado **a)**, desarrolla el programa (todos los pasos hasta la implementación en Java) que nos piden tomando los datos de las llamadas realizadas del fichero de texto **LLAMADAS.TXT** y creando el fichero de texto de salida **TIEMPO_MAX.TXT**.

c) Què estructura o estructuras elegirías ***si los teléfonos NO fueran un máximo de 100***.
Trata de utilizar alguno de los TAD conocidos. Por cada TAD utilizado indica el tipo de sus elementos.

Formato de los ficheros

El fichero **LLAMADAS.TXT** contiene una línea por cada llamada realizada y en el ***orden en el que se produjeron***. La información de cada línea consta del teléfono al que se llama (9 cifras), extensión de la que parte la llamada (3 cifras) y duración en minutos. En cada línea los números vienen separados entre sÌ por un espacio en blanco.

```
948750000 314 2
943218000 235 10
948750000 314 3
943007789 135 3
948750000 093 5
944700988 314 12
943218000 211 7
948750000 122 30
...
```

El fichero **TIEMPO_MAX.TXT** contiene información de todas las llamadas realizadas al número al que más se llama basándose en el tiempo total de conversación. La información de las llamadas aparece en el ***orden en el que se produjeron***. La cabecera del fichero incluye el número del teléfono al que se dirigen las llamadas y el tiempo total. Cada línea incluye la extensión desde la que se llama y el tiempo de esa conversación.

```
número: 948750000
tiempo total: 2000

Lista de llamadas:
314 2
314 3
093 5
122 30
...
```

(S98) Nos dan la siguiente especificación del paquete listas_ordenadas, cuyos elementos son enteros, donde la representación elegida consiste de 2 pilas:

```
public class ListasOrdenadas extends Object {
private Stack pila1=new Stack();
private Stack pila2=new Stack();

-- Inicializa la listas
public void inicializar()
-- Inserta en la lista ordenada el entero elem.
-- Si elem ya estaba en la lista lo inserta igualmente.
public void insertarOrdenado(int elem)
-- Elimina de la lista ordenada todas las apariciones del entero elem.
-- Si el elemento no se encuentra en la lista, Èsta permanece intacta.
public void eliminarOrdenado(int elem)
-- Precondición: la lista no será vacía.
-- Devuelve el primer entero de la lista.
public int primero ()
-- Precondición: la lista no será vacía.
-- Devuelve una lista ordenada como la de entrada , pero sin el primer entero.
public void resto ()
-- Devuelve true si lista está vacía y falso en caso contrario.
public boolean esVacia()
-- Visualiza el contenido de la lista
public void visualizar ()
```

Se pide

a) Haz un **dibujo** que represente los valores almacenados en pila1 y pila2 cuando la lista contenga los enteros 1,2,3,3,4.
b) Según tu idea de implementación, ¿podría darse el caso de que la lista contuviera los enteros 1,2,3,3,4, pero que éstos estuvieran dispuestos de una manera diferente a la que has dibujado en el apartado a? ¿Por qué?
c) **Diseña e implementa** los métodos esVacia y eliminar_ordenado.
d) Comparamos ahora esta representación del tipo lista_ordenada, con una lista ligada simple. Indica que operaciones tienen diferente orden de complejidad en una y otra implementación, y cuál es éste.
e) Cuál de las dos representaciones mencionadas elegirías.

Soluciones a los ejercicios de uso de TAD

Para la resolución de todos los ejercicios se ha utilizado una Token definida por el usuario para facilitar la lectura de los datos del fichero. La interfaz pública de esta clase es la siguiente:

```
public class Token {

 //El método constructor, como parámetro un String con el nombre del
   fichero que se quiere gestionar
  public Token(String file) {}

  // Devuelve cierto si quedan mas líneas por leer en el fichero, o falso en
    caso contrario
  public boolean hasMoreTokens() {}

  //Devuelve un objeto de tipo Data con la información contenida en una
    línea. El contenido de este objeto dependerá del la aplicación
  public Data getTokens() {}
}
```

Como ejemplo, en el ejercicio J94, la clase Data, tendría la siguiente especificación:

```
public class Data extends Object {

private int player1;
private int player2;
private int quantity;

  //Constructor
  public Data(int pPly1,int pPly2,int pQ) {}

  //Devuelve el jugador 1
  public int getPlayer1() {}

  //Devuelve el jugador 2
  public int getPlayer2() {}

  //Devuelve la cantidad de billetes transferidos
  public int getQuantity() {}
  }
```

En cada ejercicio aparecerá el programa principal (*main*) así como la clase a la que hace referencia dicho programa con la inplementación de los métodos.