Estructuras de Datos y Algoritmos

Grados de la Facultad de Informática (UCM)

Examen SEGUNDO CUATRIMESTRE, 2 de julio de 2019

Normas de realización del examen

1. Debes desarrollar e implementar soluciones para cada uno de los ejercicios, probarlas y entregarlas en el juez automático accesible en la dirección `http://exacrc`.

2. En el juez te identificarás con el nombre de usuario y contraseña que has recibido al comienzo del examen.

3. Escribe tu **nombre y apellidos** en un comentario en la primera línea de cada fichero que subas al juez.

4. Del enlace **Material para descargar** dentro del juez puedes descargar un archivo comprimido que contiene material que puedes utilizar para la realización del examen (transparencias de clase, implementación de las estructuras de datos, una plantilla de código fuente y ficheros de texto con los casos de prueba de cada ejercicio del enunciado).

5. Los ficheros con las implementaciones de las estructuras de datos están instalados en el juez, por lo que no es necesario subirlos como parte de tu solución (y conviene no hacerlo).

6. Los ejercicios están identificados con el nombre del tema de la asignatura en el que habrían aparecido si hubieran sido propuestos como ejercicios durante el curso. Para obtener la máxima puntuación, las soluciones deberán seguir los criterios exigidos a los ejercicios de ese tema durante el curso.

7. Tus soluciones serán evaluadas por el profesor independientemente del veredicto del juez automático. Para ello, el profesor tendrá en cuenta **exclusivamente** el último envío que hayas realizado de cada ejercicio.

8. Al terminar el examen, dirígete al puesto del profesor y rellena con tus datos la hoja de firmas que él tendrá. Muéstrale tu documento de identificación.

Primero resuelve el problema. Entonces, escribe el código.

— John Johnson

Comentar el código es como limpiar el cuarto de baño;
nadie quiere hacerlo, pero el resultado es siempre
una experiencia más agradable para uno mismo y sus invitados.

— Ryan Campbell

Ejercicio 1. Tipos de datos lineales (2 puntos)

Entremezclar dos listas consiste en formar otra tomando alternativamente elementos de cada una de esas listas. Por ejemplo, si entremezclamos la lista 1 3 5 7 con la lista 2 4 6 8, obtenemos la lista 1 2 3 4 5 6 7 8. Cuando una de las dos listas se queda vacía, se cogen todos los elementos de la otra.

Queremos extender la clase `deque` (lo importante para el ejercicio es que internamente la cola está representada con una lista enlazada circular doble de nodos dinámicos con nodo fantasma) con una nueva operación `entremezclar` que mezcle una lista enlazada doble (la del objeto *this*) con otra (recibida como argumento), comenzando a coger elementos de la primera lista. La lista recibida como argumento pasará a ser vacía.

Para resolver este ejercicio no se puede crear ni destruir memoria dinámica (hacer `new` o `delete`) al entremezclar, ni tampoco modificar los valores almacenados en los nodos de las listas enlazadas.

Entrada

La entrada consta de una serie de casos de prueba. En la primera línea aparece el número de casos de prueba que vendrán a continuación.

Cada caso se muestra en cuatro líneas. La primera contiene el número N de elementos de la lista principal (un número entre 0 y 100.000). En la segunda línea se muestran esos N elementos, números entre 1 y 1.000.000, separados por espacios. La tercera línea contiene el número M de elementos de la segunda lista (un número entre 0 y 100.000) y la cuarta línea contiene esos M elementos, números entre 1 y 1.000.000.

Salida

Para cada caso de prueba se escribirá una línea que contendrá los elementos de la lista modificada tras entremezclarla con los elementos de la segunda lista.

Entrada de ejemplo

```
3
4
1 3 5 7
4
2 4 6 8
3
7 3 9
0

2
2 1
4
5 3 4 6
```

Salida de ejemplo

```
1 2 3 4 5 6 7 8
7 3 9
2 5 1 3 4 6
```

Ejercicio 2. Aplicaciones de tipos abstractos de datos (3 puntos)

Queremos gestionar la adquisición y venta de productos por parte de una tienda. De vez en cuando la tienda recibe nuevos productos (identificados por un código, que es un `string` sin espacios) y señalados con una fecha. Las unidades del producto adquirido se guardan en el almacén, salvo que haya clientes que hubieran intentado comprar ese determinado producto cuando no había existencias y se hubieran colocado en la lista de espera. En ese caso, los clientes son servidos en riguroso orden de llegada. También puede haber venta de productos en existencia. En ese caso, se venden siempre las unidades con una fecha menor.

Para ello se desea disponer de las siguientes operaciones:

- constructora: al comienzo el almacén está vacío y no hay clientes en espera.
- `adquirir(COD, F, CANT)`: gestiona la adquisición de `CANT` unidades del producto `COD` con fecha `F`. Si hubiera clientes esperando la llegada de este producto, devuelve los identificadores de los clientes que han podido ser servidos, en el orden en que hicieron la petición. El resto de unidades (si las hay) se guardan en el almacén.
- `vender(COD, CLI)`: gestiona la venta de una unidad del producto `COD` al cliente `CLI` (un `string` sin espacios). Si hay existencias, la operación devuelve `true` y la fecha del producto vendido (la menor entre las disponibles). Si no hay existencias, devuelve `false` y añade al cliente a la lista de espera de este producto (un cliente puede aparecer varias veces en la lista de espera).
- `cuantos(COD)`: devuelve cuántas unidades tiene la tienda del producto `COD` (independientemente de la fecha).
- `hay_esperando(COD)`: indica si hay clientes en la lista de espera del producto `COD`.

Selecciona un tipo de datos adecuado para representar la información. Puedes utilizar el tipo `fecha` que os proporcionamos. En la cabecera de cada función debe indicarse el coste de la misma. Los métodos del TAD no deben mostrar nada por pantalla. El manejo de la entrada y salida de datos se realiza en la función `resuelveCaso` que os proporcionamos.

Entrada

La entrada consta de una serie de casos de prueba. Cada caso consiste en una serie de líneas donde se muestran las operaciones a realizar, sobre una tienda inicialmente vacía: el nombre de la operación seguido de sus argumentos. Cada caso termina con una línea con la palabra `FIN`.

Salida

Para cada caso de prueba, se escribirá una línea por operación de la siguiente manera:

- `adquirir`, muestra `PRODUCTO ADQUIRIDO` seguido de los códigos de los clientes que estuvieran en la lista de espera (si los había) y hayan podido llevarse una unidad;
- `vender`, si hay existencias en ese momento se muestra `VENDIDO` y la fecha del producto vendido; si no, se muestra `EN ESPERA`;
- `cuantos`, muestra el número devuelto por la operación;
- `hay_esperando`, si hay clientes esperando a que llegue ese producto escribe `SI`, y en caso contrario escribe `NO`.

Cada caso termina con una línea con tres guiones (`---`).

Entrada de ejemplo

```
vender lapiz Ana
adquirir lapiz 10/06/19 3
vender lapiz Pedro
adquirir boli 20/06/19 3
adquirir boli 15/06/19 2
vender boli Pedro
vender boli Luis
vender boli Marta
cuantos boli
hay_esperando boli
FIN
vender boli Ana
hay_esperando boli
hay_esperando lapiz
vender boli Pedro
vender boli Luis
adquirir boli 20/06/19 2
cuantos boli
hay_esperando boli
FIN
```

Salida de ejemplo

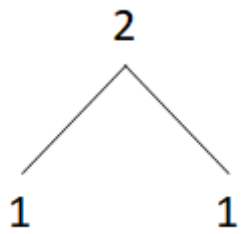
```
EN ESPERA
PRODUCTO ADQUIRIDO Ana
VENDIDO 10/06/19
PRODUCTO ADQUIRIDO
PRODUCTO ADQUIRIDO
VENDIDO 15/06/19
VENDIDO 15/06/19
VENDIDO 20/06/19
2
NO
---
EN ESPERA
SI
NO
EN ESPERA
EN ESPERA
PRODUCTO ADQUIRIDO Ana Pedro
0
SI
---
```

Estructuras de Datos y Algoritmos

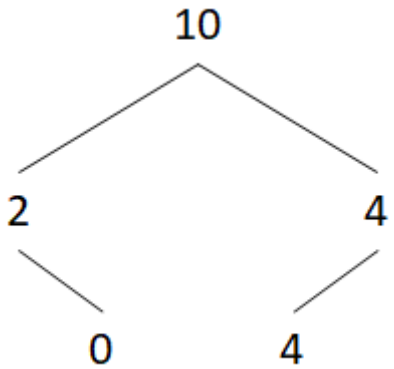
Grados en Ingeniería Informática

Examen Segundo Cuatrimestre, 2 de julio de 2019. Grupos B y D

1. (4 puntos) Un nodo interno de un árbol binario de enteros se dice que es *sumativo* si su valor es igual a la suma de los valores de sus descendientes. A modo de ejemplo, un árbol vacío o un árbol hoja tendrán 0 nodos internos sumativos. El árbol



tendrá 1 nodo interno sumativo, y el árbol



tendrá 2 nodos internos sumativos. **Implementa** el subprograma

```
int numNodosSumativos (Arbin<int> a)
```

que devuelva el número de nodos internos sumativos que tiene el árbol dado. **Determina de forma justificada la complejidad** de dicho subprograma.

2. (6 puntos) Nos han encargado implementar un módulo para la gestión de surtidores en una gasolinera. La gasolinera tiene surtidores, cada uno de los cuales puede servir distintos tipos de combustible. Cuando un vehículo llega a la gasolinera, se pone en la cola de espera de alguno de los surtidores y reposta combustible (los detalles de pagos no se cubrirán en este módulo). Así mismo, en cualquier momento un vehículo puede abandonar la cola de espera y seguir su camino. La implementación de este módulo se llevará a cabo como un `TAD GestorSurtidores` con las siguientes operaciones:

- `crea`: creación de un nuevo gestor de surtidores.
- `an_surtidor(id_surtidor)`: se añade un nuevo surtidor con identificador `id_surtidor`. Si el surtidor ya existe en el sistema, se señala un error (excepción `ESurtidorDuplicado`).
- `carga(id_surtidor, tipo_combustible, num_litros)`: carga `num_litros` de combustible `tipo_combustible` en el surtidor `id_surtidor`. Si el surtidor no existe en el sistema, se señala un error (excepción `ESurtidorNoExiste`).
- `pon_en_espera(id_vehículo, id_surtidor)`: pone en espera el vehículo `id_vehículo` en el surtidor `id_surtidor`. Si el vehículo ya está esperando en otro surtidor, o bien `id_surtidor` no existe, se señala un error (excepción `ELlegadaVehiculo`).
- `vende(id_surtidor,tipo_combustible,num_litros)` $\rightarrow$ `resul`: realiza una venta de `num_litros` de combustible `tipo_combustible` al primer vehículo que está esperando en el surtidor `id_surtidor`. Tras la venta, se actualiza la reserva de combustible del surtidor y el vehículo abandona la gasolinera.

  Una vez realizada la venta, la operación devuelve un objeto que contiene los siguientes campos: (i) el identificador del vehículo al que se ha realizado la venta; y (ii) la cantidad de combustible de tipo `tipo_combustible` que queda aún en el surtidor.

  Para que esta operación puede realizarse: (i) el surtidor debe existir; (ii) debe haber vehículos esperando para abastecerse en el mismo; (iii) el surtidor debe disponer de la cantidad suficiente de combustible solicitado; y (iv) `num_litros` debe ser mayor que 0. Si se viola alguna de estas condiciones, la operación señala un error (excepción `EErrorVenta`).
- `abandona(id_vehiculo)`: el vehículo `id_vehiculo` abandona la cola en la que está esperando y sale de la gasolinera. Si el vehículo no está esperando en ninguna cola, la operación no tiene efecto.

Dado que este módulo es un sistema crítico, debe **elegirse una representación** que permita implementar lo más eficientemente posible las operaciones pedidas, así como **llevar a cabo dicha implementación**. Para cada operación debe **indicarse justificadamente su complejidad**.