Estructuras de Datos y Algoritmos
Grado en Ingeniería Informática
Universidad de Zaragoza – curso 2023-2024

Sesión de problemas 8
Ejercicios de repaso

Objetivos

- Ejercicios de repaso de especificación.
- Ejercicios de repaso de diseño de estructuras de datos.
- Ejercicios de repaso de representación de datos e implementación (en pseudocódigo) de distintas operaciones sobre estructuras de datos vistas en la asignatura.

1. Ejercicios de especificación

1.1. TAD `EspacioMatic`

Examen 06-09-2022. Eres un prestigioso diseñador de videojuegos, y estás trabajando ahora en uno llamado `EspacioMatic`, un sistema en el que podrá haber un gran número de naves espaciales, cada una de ellas equipada con distintos módulos (motores, cabinas, escudos, láseres, etcétera). Cada nave tendrá un identificador único. Necesitarás especificar, como poco, las siguientes operaciones:

- `nuevaNave`, para añadir una nueva nave espacial (vacía, es decir, sin módulos) al sistema;
- `equipaNave`, para añadir, a una nave dada, un módulo (una cadena como "motor", "cabina", "láser", etcétera) y un nivel de funcionalidad para ese módulo (un entero ≥ 1), de forma que, si esa nave ya tenía ese módulo, se suma el nuevo nivel al anterior (esto permitirá reparar módulos de naves);
- `estropeaNave`: dada una nave, y un nombre de módulo, resta 1 al nivel de ese módulo en esa nave (asumiendo que tuviese un nivel > 0);

Distribuido bajo licencia CC BY-NC-SA 4.0. (© Profesorado del DIIS)
`https://creativecommons.org/licenses/by-nc-sa/4.0/deed.es`

- `navesDefectuosas`: para obtener un listado de identificadores de naves que tienen uno o más módulos completamente estropeados (con un nivel de 0);
- `módulosNave`: para obtener un listado, ordenado alfabéticamente, con los nombres y estado de cada uno de los módulos que tiene equipados una cierta nave (tengan el nivel que tengan).

Se debe escribir una especificación para el TAD `EspacioMatic` con al menos las operaciones descritas en el enunciado, así como cualquier otro TAD que se considere necesario.

1.2. TAD `Librería`

Examen 01-02-2022. Está popularizándose un modelo editorial flexible, consistente básicamente en ofrecer una `Librería` a la que pueden suscribirse tanto autores como lectores, y en la que pueden vender o comprar el acceso a obras para su lectura. Pretendemos desarrollar un sistema que permita gestionar la `Librería`, que se describe a continuación.

Los autores suscritos contarán con una descripción y un identificador único de escritor, y podrán ofrecer en dicha librería cero, una o más obras suyas, cada una a un determinado precio. Los lectores suscritos contarán con un identificador único de lector, un saldo de dinero disponible (número real), que podrán usar para comprar el acceso a aquellas obras que deseen, podrán recargar su saldo, y podrán votar o asignar una valoración (valor entero de 0 a 10, mejor cuanto mayor) a cada obra que hayan comprado y que no hayan valorado previamente. Cada obra contará con un identificador único en la librería, un autor único y que debe estar registrado en la librería, su título, una descripción, su precio, el número total de votos recibidos, y la suma total de las puntuaciones recibidas en los votos. Por simplificar, consideraremos que: los autores no podrán comprar obras y los lectores no podrán vender obras; las obras pueden ofrecerse gratis pero no por precio negativo; la compra de una obra deberá decrementar el saldo disponible del lector en la misma cantidad que el precio fijado para la obra; no está limitado el número de obras que puede ofrecer un autor ni el número de lectores que pueden comprar una obra; no se permite la compra sin saldo suficiente; no se permite devolver o deshacer compras de obras, ni eliminar o retirar obras; y tampoco se permite el préstamo de obras entre lectores.

Se debe escribir una especificación para el TAD `Librería` con, al menos, las operaciones:

- `registrarLector`: para añadir, si es posible, un nuevo lector al sistema, dado su identificador y su saldo inicial.
- `registrarAutor`: para añadir, si es posible, un nuevo autor al sistema, dado su identificador y su descripción.
- `registrarObra`: para añadir, si es posible, una nueva obra al sistema, dado su identificador de obra, el de su autor, su título, descripción, y su precio de venta a los lectores.

© CC BY-NC-SA 4.0 Profesorado del DIIS (Universidad de Zaragoza, España)

- `comprarObra`: para registrar, si es posible, que un lector compra el acceso a una obra, dados sus respectivos identificadores.
- `votarObra`: para registrar, si es posible, que un lector asigna su valoración personal a una obra, dados sus respectivos identificadores, y un valor de 0 a 10 expresando su valoración.
- `recargarSaldo`: para registrar, si es posible, que un lector incrementa su saldo disponible con una cantidad de dinero dada.

1.3. TAD `Rastreo`

Examen 08-09-2021. Con intención de aportar nuestro granito de arena en la gestión del seguimiento de la pandemia de COVID-19, se propone diseñar un sistema que ayude a los rastreadores a gestionar la información sobre los casos *abiertos* a los que se debe hacer un seguimiento: tanto aquellos que han dado positivo a una prueba de infección activa, como a aquellos que han sido identificados como contacto estrecho de algún caso positivo y que también deben ser testados para comprobar si resultan haber sido infectados. Todos los casos *abiertos*, ya sean de *positivo* o de *contacto*, deberán ser testados[1] para detectar la infección activa por COVID-19, y serán considerados casos a los que los rastreadores deben hacer un seguimiento hasta que se les registren dos resultados negativos cronológicamente consecutivos (sin un positivo entre ellos) a pruebas de detección de infección activa, tras lo cual el sistema pasará a retener la información de dichos casos como casos *archivados*. (Nota: no se pretende reconstruir las cadenas de contagios)

De cada caso registrado en el sistema se mantendrá la siguiente información: un identificador de caso (único, cadena), su información personal (por simplificar el ejercicio lo consideraremos como una cadena que incluya toda su información: nombre, apellidos, teléfono, domicilio, etc.), historial (cadena), estado (*abierto* o *archivado*), tipo de caso (*positivo* o *contacto*), y hasta tres fechas: la de su último resultado positivo, y las del primer y segundo resultados negativos sin resultado positivo entre ellas, y por tanto estas dos últimas serán las de las pruebas que deban usarse para justificar el archivo del caso. Para comparar fechas se usará siempre su orden cronológico. Todo caso positivo tendrá necesariamente una fecha con su último resultado positivo de infección activa. Todo caso archivado deberá tener registradas las fechas de sus dos últimos resultados negativos (consecutivos) que han justificado su archivo. Para registrar el caso de un contacto, deberá comprobarse que esté relacionado con un caso registrado positivo de infección por COVID-19 (este abierto o no). Si un caso registrado en el sistema por ser contacto estrecho obtiene un resultado positivo de infección activa por COVID-19, pasará a considerarse como caso positivo.

El sistema descrito deberá ofrecer, al menos, las siguientes operaciones:

- `añadirPositivo`: que dado un identificador de caso, y la fecha de un resultado

[1] A criterio de los especialistas médicos, para el sistema solo debemos preocuparnos de que pueda gestionar adecuadamente la información de los resultados de dichas pruebas.

© CC BY-NC-SA 4.0 Profesorado del DIIS (Universidad de Zaragoza, España)

positivo, además de su información personal e historial, registre adecuadamente en el sistema un caso abierto de positivo de infección activa por COVID-19.

- `añadirContacto`: que dado el identificador de un caso positivo registrado previamente, y además un identificador para un nuevo caso, su información personal e historial, registre adecuadamente en el sistema un caso abierto por ser contacto estrecho del caso positivo previamente registrado.
- `registrarNegativo`: que dado el identificador de un caso y una fecha, registre adecuadamente en dicho caso que en esa fecha ha obtenido un resultado negativo en una prueba de infección activa por COVID-19.
- `últimoNegativo`: que dado un identificador de caso, devolverá si es posible, la fecha de su último resultado negativo registrado.
- `casoActivo`: que dado un identificador de caso, si es posible, devolverá un booleano indicando si corresponde a un caso activo o a uno archivado.

Se pide:

Especificar el TAD `Rastreo` con las características y, al menos, las operaciones descritas en el enunciado y con otras que se consideren necesarias. Se puede considerar ya disponible un TAD `fechas` con la semántica obvia (día y hora) y las operaciones que se consideren necesarias.

1.4. TAD `MultaMatic`

Examen 02-02-2021. Pretendemos desarrollar `MultaMatic`, el nuevo sistema de gestión de multas de tráfico por exceso de velocidad. La red de carreteras contiene tramos vigilados en los que se coloca una cámara al principio del tramo y otra al final (se considera la marcha de vehículos en un único sentido, del principio del tramo al final). Cada vez que un coche pasa frente a una cámara, se toma una foto de su matrícula y se apunta el momento en que pasó; si el tiempo trascurrido entre la foto del comienzo y la del final del tramo es demasiado breve, se le pone automáticamente una multa. Para simplificar, asumiremos que los tramos no comparten cámaras ni se solapan entre sí. Cada tramo tiene un identificador único. Cada cámara tiene un identificador único. Se debe escribir una especificación para el TAD `MultaMatic` con, al menos, las operaciones:

- `insertaTramo`: para añadir un nuevo tramo al sistema, dados los identificadores de tramo y de sus cámaras inicial y final, y el número mínimo de segundos que deben transcurrir entre las fotos de comienzo y final para no recibir multa;
- `fotoEntrada`: para registrar que un coche entra en un tramo vigilado, dados el identificador de la cámara, la matrícula del coche, y el instante actual (expresado en segundos transcurridos desde el 1 de enero de 2001);

© CC BY-NC-SA 4.0 Profesorado del DIIS (Universidad de Zaragoza, España)

- `fotoSalida`: para registrar que un coche sale de un tramo vigilado, dados el identificador de la cámara, la matrícula del coche, y el instante actual; si el coche ha ido demasiado rápido en el tramo, se registra una multa más al coche;
- `multasPorMatrícula`: para conocer el número de multas registradas para un coche;
- `multasPorTramos`: para obtener un listado con las matrículas de los coches multados en un determinado tramo; si un coche ha sido multado varias veces, su matrícula aparecerá igual número de veces.

1.5. TAD `Clientes`

Examen 07-09-2020. Se quiere almacenar los números de teléfono de los clientes de una compañía. Cada cliente tiene un identificador único, `IDcliente`. Un cliente puede tener varios números de teléfono (no más de 10), aunque puede quedarse sin ningún teléfono. Un mismo número de teléfono puede ser compartido por varios clientes (no más de 10). Se precisan, como mínimo, las siguientes operaciones:

- Insertar un par (`IDcliente c`, número de teléfono `n`); es decir, registrar, si es posible, que se puede contactar con el cliente con identificador `c` utilizando el número de teléfono `n`. Si el cliente o el número de teléfono no existían, los incorpora al sistema.
- Borrar un par (`IDcliente c`, número de teléfono `n`); es decir, registrar, si es posible, que el cliente con identificador `c` ya no utilizará el número `n`. Si el número no es usado por ningún otro cliente, también se elimina del sistema.
- Borrar un cliente `c`; es decir, eliminar, si es posible, ese cliente del sistema y todos los números que utilizase (con las mismas reglas que la operación anterior).
- Buscar un cliente con identificador `c`, y obtener el listado de teléfonos que usa.
- Buscar un número de teléfono `n`, y obtener el listado de identificadores de clientes que lo utilizan.

Todas las operaciones deben poder ejecutarse con coste $\mathcal{O}(\log C)$ en el caso peor, siendo C el número de clientes registrados (nótese que como cada cliente usa como mucho 10 teléfonos, el número de teléfonos registrados es como mucho $10C$).

Se pide:

Especificar el TAD `Clientes` con, al menos, las operaciones descritas en el enunciado y con otras que se consideren necesarias.

© CC BY-NC-SA 4.0 Profesorado del DIIS (Universidad de Zaragoza, España)

1.6. TAD `ComunidadActiva`

Examen 31-01-2020. La Universidad de Zaragoza se ha adherido a los *Objetivos de Desarrollo Sostenible* (ODS), reflejados en la Agenda 2030 de las Naciones Unidas, que buscan promover la implicación y participación activa para alcanzar los objetivos identificados, sumando los esfuerzos de gobiernos y de todo tipo de entidades, organizaciones, empresas, comunidades y de la ciudadanía en general. Los ODS identifican 17 objetivos, dividiéndose cada uno de ellos en varias metas específicas a alcanzar, 169 en total, pero no queremos descartar que puedan identificarse aún algunas pocas metas más.

Para promover y facilitar la participación de todos los miembros de una comunidad, se desea diseñar un sistema que permita gestionar las metas identificadas, y las acciones a desarrollar propuestas por participantes de la comunidad. Para ello, se deberá gestionar información de las metas perseguidas (identificadas cada una de ellas mediante una cadena, y acompañadas por una descripción textual de la meta), gestionar la información de los participantes en la comunidad (identificados mediante su DNI, y de los cuales se registrará también su nombre, apellidos, y fecha de nacimiento), y gestionar la información de las acciones propuestas en la comunidad. Toda acción se identificará mediante una cadena única, contará con una descripción textual, deberá ser propuesta por un participante de la comunidad, y deberá contribuir a avanzar significativamente para alcanzar al menos una de las metas perseguidas por la comunidad (se puede considerar que ninguna acción puede contribuir significativamente a más de 10 metas distintas). Cualquier participante de la comunidad podrá proponer todas las acciones que desee. Las operaciones que deberá ofrecer el sistema descrito, llamémosle `ComunidadActiva`, deberán ser al menos:

- `registrarMeta`: que añade, si es posible, la información de una nueva meta en la comunidad.
- `registrarParticipante`: que añade, si es posible, la información de un nuevo participante en la comunidad.
- `registrarAcción`: que añade, si es posible, la información de una nueva acción propuesta en la comunidad.
- `registrarMetaEnAcción`: que añade, si es posible, la información de que cierta acción `A` contribuye significativamente a alcanzar cierta meta `M`.
- `metaPocoPerseguida`: que devolverá, si es posible, la identificación de una meta con mínimo número de acciones propuestas para alcanzarla.

Se pide:

Especificar el TAD `ComunidadActiva` con al menos las operaciones descritas en el enunciado, y con otras que se consideren necesarias. Se puede considerar ya disponible un TAD `Fecha` con la semántica obvia (día, mes y año) y con las operaciones que se consideren necesarias.

© CC BY-NC-SA 4.0 Profesorado del DIIS (Universidad de Zaragoza, España)

1.7. TAD `GestiónPolicial`

Examen 09-09-2019. La Dirección Provincial de Policía de una provincia española necesita una aplicación de gestión de su larga lista de personas fichadas (por sus delitos cometidos en el pasado) residentes en la provincia, con objeto de facilitar tanto las investigaciones en curso como el saber de qué informar a la prensa local cuando sea necesario. Para cada persona fichada, se cuenta con sus datos personales y policiales: nombre, alias, DNI, sexo, fecha de nacimiento, municipio de residencia, si está en prisión o no, y descripción de delitos cometidos. Según datos del Instituto Nacional de Estadística, la provincia española con mayor número de municipios es Burgos (con 371).

Entre las operaciones más frecuentes que la Policía realiza están las siguientes:

- Alta/modificación de la ficha de una persona.
- Búsqueda de la ficha de una persona a partir de su DNI para obtener sus datos.
- Calcular el grado de peligrosidad de un municipio, definido como el número de personas fichadas con residencia en él (y que no están en prisión).
- Obtener la lista con los DNIs de las personas fichadas de un municipio (y que no están en prisión).

Se pide:

Especificar el TAD `GestiónPolicial` con las operaciones mencionadas y otras que se consideren imprescindibles o de especial interés, así como nuevos TADs que se consideren imprescindibles.

1.8. TAD `RedSocial`

Examen 28-01-2019. Se desea diseñar un sistema similar a una red social, tipo Twitter, en la que cada usuario pueda "seguir" a otros usuarios de su elección, y acceder fácilmente a los últimos mensajes publicados por dichos usuarios, además de publicar sus propios mensajes. Cada usuario será identificado mediante un nombre único (real o no), y mientras pertenezca a la red social podrá publicar cualquier número de mensajes y en cualquier momento. Para cada mensaje se conservará el texto del mensaje, y el instante (día y hora) de su publicación. Cada usuario podrá añadir, o quitar, a otro usuario al grupo de usuarios que considera para sí de especial interés, diremos que sigue a dichos usuarios. Además, cada usuario podrá consultar fácilmente un número arbitrariamente grande de los últimos mensajes publicados por todos aquellos usuarios a los que sigue. Las operaciones que deberá ofrecer el sistema descrito, llamémosle `RedSocial`, deberán ser al menos:

- `añadirUsuario`: que añade, si es posible, la información de un nuevo usuario al sistema.

© CC BY-NC-SA 4.0 Profesorado del DIIS (Universidad de Zaragoza, España)

- `eliminarUsuario`: que dado el nombre de un usuario elimina, si es posible, toda su información del sistema.
- `seguirA`: que añade, si es posible, la información de que cierto usuario `A` sigue a otro usuario dado `B`.
- `dejarDeSeguir`: que elimina, si es posible, la información de que cierto usuario `A` sigue a otro usuario dado `B`.
- `publicar`: que añade, si es posible, la información de un nuevo mensaje de un usuario dado.
- `últimasPublicaciones`: que dado un valor `k` devuelve, si es posible, la información de los `k` últimos mensajes publicados por los usuarios a los que sigue un usuario dado (`k` mensajes en total, no `k` de cada uno).

Se pide:

Especificar el TAD `RedSocial` con al menos las operaciones descritas en el enunciado, y con otras que se consideren necesarias. Se puede considerar ya disponible un TAD `Instante` con la semántica obvia (día y hora) y las operaciones que se consideren necesarias.

2. Ejercicios de implementación

2.1. Implementación del procedimiento `procesa`

Examen 11-01-2023. Escribir el siguiente procedimiento en pseudocódigo:

```
procedimiento procesa(e/s L: lista; sal Limpares: lista)
```

donde el tipo `lista` está definido como:

```
tipos
    lista = ↑nodo;
     nodo = registro
                dato: entero;
                sig: lista
            freg
```

que, de la forma más eficiente posible, modifica la lista `L` y además devuelve la lista `Limpares` de la siguiente manera:

- cada dato entero almacenado en L que sea número impar es eliminado de la lista `L` y añadido a la nueva lista `Limpares`, y
- cada dato entero almacenado en `L` que sea número par se duplica en la lista `L` (añadiendo el dato duplicado inmediatamente a continuación).

Ejemplo: si `L = (-7,0,-2,3,5,8,8,9,-1,4)`, se deberá devolver `L = (0,0,-2,-2,8,8,8,8,4,4)` y `Limpares = (-7,3,5,9,-1)`.

© CC BY-NC-SA 4.0 Profesorado del DIIS (Universidad de Zaragoza, España)

2.2. Implementación del procedimiento `minDrag`

Examen 06-09-2022. Dado un árbol binario en cuya raíz se encuentra situado un tesoro y cuyos nodos internos pueden contener un dragón o no contener nada, se desea tener un algoritmo que nos indique ***la hoja del árbol cuyo camino hasta la raíz tenga el menor número de dragones***. En caso de que existan varios caminos con un mínimo número de dragones, el algoritmo devolverá uno cualquiera de entre todos ellos. Para ello, los nodos del árbol binario almacenan cadenas, de acuerdo a los tipos de datos definidos a continuación:

```
tipos
          arbin = ↑nodoÁrbol;
      nodoÁrbol = registro
                      dato: cadena;
                      izq, der: arbin
                  freg
```

y de forma que:

- la raíz contiene la cadena `"tesoro"`;
- los nodos internos contienen la cadena `"dragón"` para indicar que en el nodo hay un dragón o la cadena `"libre"` para indicar que no hay dragón;
- en cada hoja se almacena una cadena que identifica a la hoja (que no puede estar repetida y no puede ser ni `"tesoro"`, ni `"dragón"`, ni `"libre"`);

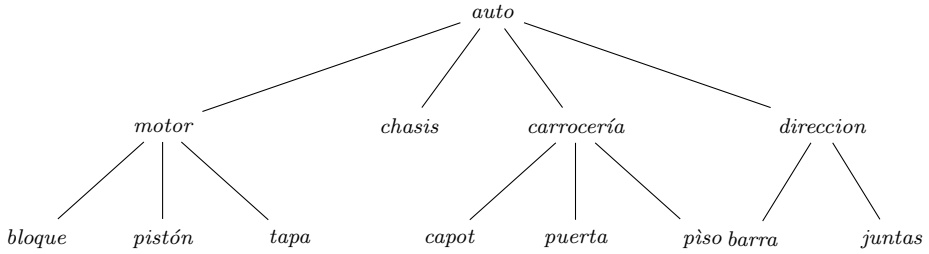
Se debe **implementar un procedimiento que recibe un árbol binario como el descrito y devuelve**: (1) la cadena identificativa de la hoja del camino seleccionado y (2) una lista encadenada que contiene los identificadores de todas las hojas del árbol en cualquier orden. Considera que el árbol tiene como mínimo un nodo raíz y un nodo hoja diferente de la raíz.

```
tipos
           lista = ↑nodoLista;
       nodoLista = registro
                       dato: cadena;
                       sig: lista
                   freg

procedimiento minDrag(ent a: arbin; sal hojaBuena: cadena; sal listaHojas: lista)
```

2.3. Implementación del procedimiento `verificar`

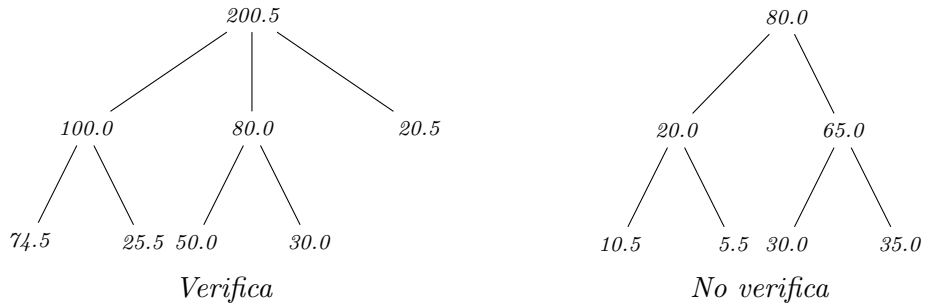
Examen 01-02-2022. En un programa de diseño asistido por computador (tipo AutoCAD) se desea mantener las piezas de un sistema (por ejemplo, un automóvil) clasificando sus partes en forma de árbol n-ario, por ejemplo:

© CC BY-NC-SA 4.0 Profesorado del DIIS (Universidad de Zaragoza, España)



Se quiere mantener en cada hoja el peso (número real, en kilos) del componente, y en los nodos interiores el peso total de todos los componentes del subárbol que cuelga de ese nodo.

Periódicamente se quiere verificar que efectivamente las etiquetas del árbol verifican esta propiedad, es decir que la etiqueta de todo nodo interno es la suma de las etiquetas de sus hijos. Así, por ejemplo, de los dos árboles siguientes, el de la izquierda verifica la condición, mientras que el de la derecha no.



Escribir un procedimiento `verifica` con la siguiente signatura:

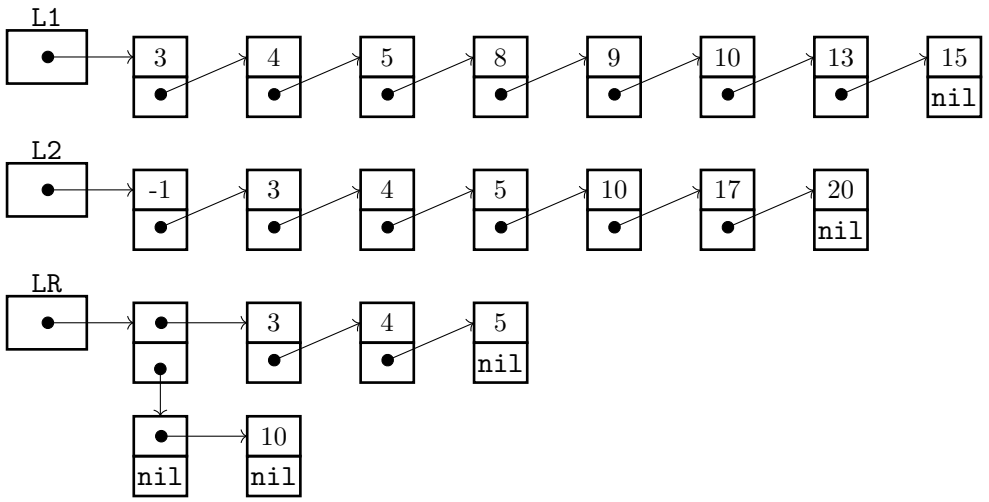
```
procedimiento verifica(ent a: árbol; sal sumaOK: booleano; sal componentes: lista)
```

lo más eficiente que sea posible, que devuelve `sumaOK = verdad` si y sólo si el árbol `a` (que es no vacío) verifica la condición y además, en caso de verificarla, devuelve en componentes una lista con todas las componentes almacenadas en las hojas del árbol (cada una con su nombre y peso); si no se verifica la condición, devuelve `sumaOK = falso` y se devuelve la lista vacía. Para ello, definir previamente las estructuras de datos necesarias para almacenar un árbol n-ario y una lista de componentes. Razonar el coste temporal del algoritmo en el caso peor.

2.4. Implementación del procedimiento `fragmentar`

Examen 08-09-2021. Implementa en pseudocódigo un procedimiento `fragmentar` que, dadas dos listas de números enteros y ordenadas `L1` y `L2`, definidas de forma similar a lo que se describe en los dibujos del ejemplo a continuación, devuelva una nueva lista de listas, `LR`, conteniendo copias de los fragmentos comunes más largos que aparecían idénticos en las listas `L1` y `L2`, y en el mismo orden en el que aparecían. Para resolver el ejercicio, define previamente en pseudocódigo los tipos de datos necesarios.

© CC BY-NC-SA 4.0 Profesorado del DIIS (Universidad de Zaragoza, España)



2.5. Implementación del procedimiento `añadir`

Examen 02-02-2021. Almacenamos la información de los descendientes de un antepasado lejano llamado, por ejemplo, Primigenio García, en un árbol n-ario de la siguiente forma. La información de cada persona se guarda en un nodo del árbol, estando la de Primigenio guardada en la raíz. De cada persona se almacena, en el nodo correspondiente del árbol, su identificador único (una cadena). Los hijos de una persona almacenada en un cierto nodo x del árbol n-ario se guardan en los nodos hijos de x en el árbol n-ario. El árbol n-ario se almacena en memoria dinámica con la representación primogénito-siguiente hermano:

```
tipos
        árbol = ↑nodoÁrbol;
    nodoÁrbol = registro
                    ID: cadena;
                    primogénito, sigHermano: árbol
                freg
```

Queremos añadir la información (identificador) de una nueva persona, por ejemplo, Natalia García, en el árbol. Para ello, recibimos una lista encadenada con punteros que almacena los identificadores de la secuencia de ancestros de Natalia, empezando desde Primigenio y terminando por ella misma. La lista se almacena de la siguiente forma:

```
tipos
        lista = ↑nodoLista;
    nodoLista = registro
                    ID: cadena;
                    sig: lista
                freg
```

Sabemos que la secuencia de ancestros de Natalia que recibimos en la lista es correcta y se encuentra almacenada ya en el árbol, excepto el identificador de Natalia, que no está almacenado todavía, y que debemos añadir. Debe implementarse, utilizando el lenguaje pseudocódigo de la asignatura, el procedimiento correspondiente para añadir el

© CC BY-NC-SA 4.0 Profesorado del DIIS (Universidad de Zaragoza, España)

identificador del último nodo de la lista `L` (Natalia García, siguiendo con el ejemplo) en el árbol `a`:

```
procedimiento añadir(e/s a: árbol; ent L: lista)
```

2.6. Implementación del procedimiento `reemplazar`

Examen 31-01-2020. Dada una lista `L` de enteros y otras dos listas no vacías, `patrón` y `sustituta`, diseñar en pseudocódigo el procedimiento:

```
tipos
     ptNodo = ↑nodo;
       nodo = registro
                  dato: entero;
                  sig: ptNodo
              freg;
      lista = registro
                  pri: ptNodo;
                  long: entero {número de nodos de la lista}
              freg

procedimiento reemplazar(e/s L: lista; ent patrón, sustituta: lista)
```

que busca todas las apariciones de patrón como subsecuencia de `L` y las sustituye por `sustituta`.

Por ejemplo, si `L={1,2,3,4,5,1,2,3,4,5,1,2,3,4,5}`, `patrón={4,5,1}` y `sustituta={9,7,3,8}`, entonces después de ejecutar `reemplazar(L, patrón, sustituta)`, debe quedar `L={1,2,3,9,7,3,8,2,3,9`
Este procedimiento tiene un efecto equivalente a la función "reemplazar" de los editores de texto.

Se sugiere el siguiente algoritmo: se recorre la lista `L`; si la subsecuencia de `L` a partir de un cierto nodo es igual a `patrón` entonces se eliminan los elementos de `L` correspondientes y se inserta `sustituta`. Si no, se avanza en `L`.

2.7. Implementación de la función `kEquilibrado`

Examen 07-09-2020. Diseñar en pseudocódigo una función `kEquilibrado` que, dado un valor natural `K` y dado un árbol binario `A`, definido por las estructuras de datos que se muestran a continuación, devuelva verdad si y sólo si para todo nodo de `A`, la diferencia de alturas entre sus subárboles hijos es como máximo `K`. No deben modificarse dichas estructuras, ni recurrirse a utilizar otras similares, y el ejercicio debe resolverse de la forma más eficiente posible.

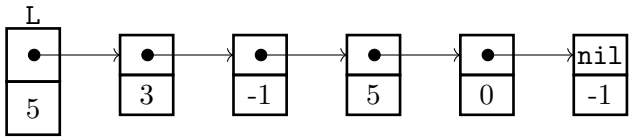
```
tipos
     arbin = ↑nodo;
      nodo = registro
               dato: elemento;
               izq, der: arbin
             freg

función kEquilibrado(K: natural; A: arbin) devuelve booleano
```

© CC BY-NC-SA 4.0 Profesorado del DIIS (Universidad de Zaragoza, España)

2.8. Implementación del procedimiento `ordenarPorMezcla`

Examen 09-09-2019. Se han definido los siguientes tipos para almacenar una lista enlazada de números enteros (junto a su longitud):

```
tipos
     ptNodo = ↑nodo;
       nodo = registro
                  sig: ptNodo
                  dato: entero;
              freg;
      lista = registro
                  pri: ptNodo;
                  long: natural
              freg
```



Diseñar el procedimiento:

```
procedimiento ordenarPorMezcla(e/s L: lista)
```

que ordene los elementos de `L`, de menor a mayor, con el método de ordenación por mezcla, que se describe a continuación:

- Si la lista contiene 0 o 1 elementos, la lista ya está ordenada.

En caso contrario:

- Partir la lista en dos listas enlazadas de (aproximadamente) igual longitud, `L1` y `L2` (sin reservar ni destruir memoria dinámica), y dejando `L` vacía.
- Recursivamente, ordenar `L1` con el mismo método.
- Recursivamente, ordenar `L2` con el mismo método.
- Mezclar las dos listas ordenadas `L1` y `L2` en una única lista `L` (vaciando al mismo tiempo las listas `L1` y `L2`). Todo ello sin reservar ni destruir memoria dinámica.

Hay que escribir el código completo (es decir, no se puede suponer que ya existen sub-algoritmos implementados).

2.9. Implementación del procedimiento `procesa`

Examen 28-01-2019. Se debe implementar en pseudocódigo el procedimiento `procesa` que reciba un árbol binario de búsqueda de estudiantes, que guarda información de una colección de estudiantes utilizando como clave de búsqueda el identificador de estudiante (`ID`), y distribuya todos los estudiantes almacenados en ese árbol en dos nuevos árboles binarios de búsqueda de estudiantes, el primero con todos los estudiantes de identificador par y el segundo con los estudiantes de identificador impar; y además devuelva una cola con todos los estudiantes del árbol de entrada, ordenados por valor creciente de `ID`.

© CC BY-NC-SA 4.0 Profesorado del DIIS (Universidad de Zaragoza, España)

```
tipos
          clase =  ↑nodoClase;
      nodoClase = registro
                      ID: entero;
                      info: InfoEstudiante;
                      izq, der: clase
                  freg;
         ptNodo = ↑nodo;
           nodo = registro
                      ID: entero;
                      info: InfoEstudiante;
                      sig: ptNodo
                  freg;
           cola = registro
                      pri, ult: ptNodo
                  freg
```

Hay que escribir el código completo (es decir, no se puede suponer que ya existen sub-algoritmos implementados). Se valorará especialmente la eficiencia de la solución.

2.10. Implementación de la función `longNivelMásPoblado` y del procedimiento `creaListaNivel`

Examen 20-01-2018. Dados los tipos de datos (en los que "elemento" es un tipo cualquiera):

a) Diseñar, en pseudocódigo, la función `longNivelMásPoblado` que reciba un árbol binario como parámetro (al que se le puede suponer un límite máximo de altura igual a 100) y devuelva el número de elementos que hay en su nivel con más elementos.

```
función longNivelMásPoblado(a: arbin) devuelve entero
```

b) Dados un árbol binario no vacío y un número entero `n` no negativo, escribir en pseudocódigo un procedimiento `creaListaNivel` que devuelva una lista encadenada que incluya todos los elementos que están en el nivel `n` del árbol (si existe).

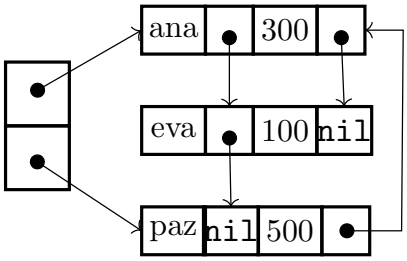
```
procedimiento creaListaNivel(ent a: arbin; ent n: entero; sal L: lista)
```

2.11. Implementación del procedimiento `inserta`

Examen 09-09-2011. Tenemos una colección de datos de alumnos que nos interesa recorrer ordenadamente según dos criterios distintos: ascendente por nombre o descendente por número de identificación personal (NIP). La solución escogida ha sido utilizar una lista enlazada en la que cada nodo tiene dos campos puntero: uno para ordenar de manera ascendente los nodos según su nombre y otro para ordenar de forma descendente los nodos según el NIP. Se pide: definir los tipos de datos necesarios (manejar el nombre en un solo `string`) y escribir un procedimiento (en pseudocódigo) que inserte un nuevo elemento en la estructura evitando que se guarden NIP's repetidos.

© CC BY-NC-SA 4.0 Profesorado del DIIS (Universidad de Zaragoza, España)

```
procedimiento inserta (e/s  l: lista; ent nombre, nip: string)
```



© CC BY-NC-SA 4.0 Profesorado del DIIS (Universidad de Zaragoza, España)