Estructuras de Datos y de la Información

Ingeniería Técnica en Informática de Gestión. Curso 2007/2008
Hoja de ejercicios número 6

Tablas

1. Se trata de enriquecer la clase `TTablaAbierta<TClave,TValor>` añadiéndole operaciones de recorrido que permitan visitar todos los nodos de la tabla sin importar el orden:

- *reinicia*, sitúa el punto de interés al "principio" de la tabla. La especificación no fija cuál ha de ser el principio de una tabla.
- *esFin*, consulta si el punto de interés está al final de la tabla. Si la tabla está vacía *esFin* ha de ser cierto.
- *actual*, devuelve una pareja `TPareja<TClave,TValor>` con el contenido del nodo situado a continuación del punto de interés. Es un error consultar por *actual* cuando *esFin* es cierto.
- *avanza*, mueve una posición hacia delante el punto de interés. Es un error ejecutar *avanza* cuando *esFin* es cierto.

Estas operaciones se deben implementar de forma que un bucle como este

```
tabla.reinicia();
while ( ! tabla.esFin() ) {
  cout << tabla.actual();
  tabla.avanza();
}
```

recorra todos los elementos almacenados en la tabla.

Indica qué cambios sería necesario realizar en la estructura de datos de la clase `TTablaAbierta<TClave,TValor>` para poder implementar estas operaciones de manera eficiente, sin que por ello se perjudique la complejidad de las otras operaciones de las tablas. Describe cómo implementarías las nuevas operaciones, qué cambios sería necesario realizar en las otras operaciones de las tablas y razona la complejidad temporal que obtendrías.

2. Se trata de desarrollar un TAD *TABLA-ORD[C :: ORD, V :: ANY]* que represente a las tablas ordenadas. Este TAD ofrecerá las siguientes operaciones:

- *Nuevo*: crea una tabla vacía.
- *Inserta*: añade una clave y un valor a una tabla. Si en la tabla ya se encontrase la clave insertada, se sustituirá el valor antiguo por el nuevo.
- *borra*: dada una clave y una tabla, elimina dicha clave, y su valor asociado, de la tabla. Si la clave no se encuentra en la tabla, se devolverá la tabla sin modificar.
- *está*: determina si una clave se encuentra en una tabla dada.
- *consulta*: dada una tabla y una clave, obtiene el valor asociado a dicha clave. Es un error consultar por el valor de una clave que no haya sido insertada previamente.
- *minClave*: obtiene la menor clave almacena en la tabla. Es un error consultar por la mínima clave de una tabla vacía.
- *esVacío*: determina si una tabla está vacía.
- *enumera*: devuelve una secuencia de pares (clave, valor) con el contenido de la tabla ordenado por claves.

Desarrolla en C++ una clase *TTablaOrd<TClave,TValor>* que implemente este TAD de forma que la complejidad de *enumera* sea $O(n)$, siendo n el número de datos almacenados en la tabla, y se perjudique lo menos posible a la complejidad de las otras operaciones. Analiza la complejidad temporal de las operaciones.

Idea: cada nodo de una lista de colisiones formará parte de dos listas, la propia lista de colisiones y una lista ordenada doblemente enlazada que contenga todos los nodos de la tabla. De esta forma, cada nodo contendrá tres punteros: el siguiente en la lista de colisiones, además de el siguiente y el anterior en la lista ordenada.

3. Se trata de desarrollar un TAD *MCJTO[A :: ORD]* que represente a los multiconjuntos. Un multiconjunto es un conjunto donde puede haber elementos repetidos. Este TAD ofrecerá las siguientes operaciones:
- *Nuevo*: crea un multiconjunto vacío.
- *Inserta*: añade un elemento al multiconjunto.
- *borra*: elimina un elemento del multiconjunto.
- *está*: determina si un elemento pertenece al multiconjunto.
- *min*: devuelve el menor elemento de entre los que contiene el multiconjunto.
- *borraMin*: elimina una aparición del menor elemento del multiconjunto.
- *esVacío*: determina si un multiconjunto está vacío.

Desarrolla en C++ una clase *TMultiCjto<TElem>* que implemente este TAD intentando minimizar la complejidad temporal de las operaciones.

Idea: representa los multiconjuntos como tablas ordenadas utilizando los elementos como claves y el número de apariciones como valor.

4. Resuelve de nuevo el *problema de las concordancias* (cfr. ejercicio 5 de la hoja 5), utilizando en lugar de un árbol de búsqueda una tabla ordenada del tipo que hemos considerado en el ejercicio 2. Analiza el tiempo de ejecución del algoritmo que obtengas.

5. Diseña un procedimiento que "limpie" una tabla cerrada, de tal manera que las posiciones borradas se eliminen y las restantes informaciones se reubiquen en la tabla. Todas las informaciones correspondientes a claves sinónimas deberán quedar situadas en posiciones consecutivas de la sucesión de pruebas, a partir de la posición primaria que corresponda.

6. Modifica la implementación de la operación de inserción en las tablas cerradas vista en clase para que cuando la tasa de ocupación supere un cierto valor fijado de antemano, se aumente la capacidad del array, se reubiquen los datos adecuadamente y se "limpie" la tabla.

Aplicaciones

Para los siguientes ejercicios has de suponer que todos los TADs vistos en clase están equipados con la operación *numElem* que devuelve el número de elementos de la estructura en cuestión.

Los árboles de búsqueda, además de la operación *recorre* que obtiene una secuencia con los valores del árbol, estarán equipados también con estas dos operaciones de recorrido:

```
TSecuenciaDinamica<TClave> recorreClave( ) const;
// Pre : true
// Post : devuelve las claves del árbol ordenadas

TSecuenciaDinamica< TPareja<TClave,TValor> > recorreClaveValor( ) const;
// Pre : true
// Post : devuelve parejas con las claves y los valores del árbol, ordenadas por clave
```

Debes considerar así mismo que las tablas (abiertas y cerradas) están equipadas con las siguientes operaciones:

```
TSecuenciaDinamica< TPareja<TClave, TValor> > enumera( ) const;
// Pre: true
// Post: Devuelve una secuencia de parejas con los elementos de la tabla

TSecuenciaDinamica<TClave> enumeraClave( ) const;
// Pre: true
// Post: Devuelve una secuencia con las claves de la tabla

TSecuenciaDinamica<TValor> enumeraValor( ) const;
// Pre: true
// Post: Devuelve una secuencia con los valores de la tabla
```

7. En este ejercicio se trata de desarrollar un sistema informático que modele el comportamiento de un *consultorio médico*. Para ello suponemos disponibles los módulos que implementan a las siguientes clases, todas ellas equipadas con operaciones de igualdad y orden:

- *TMedico* que sirve para almacenar y gestionar la información sobre un médico del consultorio.
- *TPaciente* que sirve para almacenar y gestionar la información sobre un paciente.

Nosotros nos encargaremos de la implementación de la clase *TConsultorio*, cuya misión es gestionar la información sobre los médicos y las citas de los pacientes. Esta clase ofrece las siguientes operaciones públicas:

- *Nuevo*: Genera un consultorio vacío sin ninguna información.
- *NuevoMédico*: Altera un consultorio, dando de alta a un nuevo médico que antes no figuraba en el consultorio.
- *PideConsulta*: Altera un consultorio, haciendo que un paciente se ponga a la espera para ser atendido por un médico, el cual debe estar dado de alta en el consultorio.
- *siguientePaciente*: Consulta el paciente a quien le toca el turno para ser atendido por un médico; éste debe estar dado de alta, y debe tener algún paciente que le haya pedido consulta.
- *atiendeConsulta*: Modifica un consultorio, eliminando el paciente al que le toque el turno para ser atendido por un médico; éste debe estar dado de alta, y debe tener algún paciente que le haya pedido consulta.
- *tienePacientes*: Reconoce si hay o no pacientes a la espera de ser atendidos por un médico, el cual debe estar dado de alta.

Desarrolla en C++ una implementación de la clase *TConsultorio* basada en otros TADs conocidos, optimizando la complejidad temporal de las operaciones.

8. En este ejercicio se trata de desarrollar un sistema informático que modele el comportamiento de un *mercado de trabajo* simplificado, donde las personas pueden ser contratadas y despedidas por empresas. Para ello suponemos disponibles los módulos que implementan a las siguientes clases, todas ellas equipadas con operaciones de igualdad y orden:

- *TPersona* que sirve para almacenar y gestionar la información sobre una persona.
- *TEmpresa* que sirve para almacenar y gestionar la información sobre una empresa.

Nosotros nos encargaremos de la implementación de la clase *TMercado*, cuya misión es gestionar la información sobre el mercado de trabajo. Esta clase ofrece las siguientes operaciones públicas:

- *Nuevo*: Genera un mercado vacío, sin ninguna información.
- *Contrata*: Altera un mercado, efectuando la contratación de cierta persona como empleado de cierta empresa.
- *despide*: Altera un mercado, efectuando el despido de cierta persona que era antes empleado de cierta empresa.
- *empleados*: Consulta los empleados de una empresa, devolviendo el resultado como secuencia ordenada de personas.
- *esEmpleado*: Averigua si es cierto o no que una persona dada es empleado de una empresa dada.
- *esPluriempleado*: Averigua si es cierto o no que una persona es empleado de más de una empresa.

Desarrolla en C++ una implementación de la clase *TMercado* basada en otros TADs conocidos, optimizando la complejidad temporal de las operaciones.

9. Nos han encargado desarrollar un sistema informático que se ocupe del almacenamiento y la gestión de *citas literarias*. La clase *TCultura* será la encargada de gestionar la información de una colección de citas, ofreciendo a sus clientes las siguientes operaciones públicas:

- *Nuevo*: inicializa una colección de citas vacía.
- *Inserta*: añade una cita a una colección de citas. Una cita viene dada como una cadena (*string*) y se puede suponer que no se insertarán citas repetidas.
- *elimina*: dada una secuencia de palabras, que se pueden suponer no repetidas, elimina las citas que contengan todas esas palabras.
- *consulta*: dada una secuencia de palabras, que se pueden suponer no repetidas, devuelve una secuencia sin repetición con las citas que contienen dichas palabras.
- *consultaAprox*: dada una secuencia de palabras no repetidas y un número *n* menor o igual que la longitud de la secuencia, devuelve una secuencia sin repetición con las citas que contienen al menos *n* palabras de la consulta.

Desarrolla en C++ una implementación de la clase *TCultura* basada en otros TADs conocidos, optimizando la complejidad temporal de las operaciones.

Idea: se puede utilizar una tabla donde cada palabra (clave) lleva asociada la lista de citas donde esa palabra aparece (valor). Para evitar repetir las citas se pueden utilizar punteros.

10. Nos han encargado desarrollar un sistema informático que se ocupe de la gestión de las notas de los alumnos de la Escuela de Informática de San Petersburgo. Para ello suponemos disponibles los módulos que implementan a las siguientes clases:

- *TAsignatura*, que sirve para almacenar y gestionar la información sobre una asignatura: el nombre de la asignatura, el profesor, el curso en el cual se imparte, …. Suponemos que está equipado con operaciones de igualdad y orden.
- *TAlumno*, que sirve para almacenar y gestionar la información sobre un alumno: su nombre, DNI, domicilio, …. Suponemos también que está equipado con operaciones de igualdad y orden.
- *TNota*, que sirve para representar las calificaciones obtenidas por los alumnos en las asignaturas: sin calificar, no presentado, suspenso, aprobado, notable y sobresaliente.

Nosotros nos encargaremos de implementar la clase *TTablon* que ofrece las siguientes operaciones públicas:

- *Nuevo*: inicializa un *Tablon* vacío.
- *InsertaAsignatura*: añade a un *Tablon* una asignatura, junto con la lista –o secuencia– de los alumnos matriculados en ella.
- *Califica*: añade a un *Tablón* la información sobre la nota de un alumno en una asignatura determinada.
- *listaNotas*: devuelve una secuencia ordenada con los alumnos que han aprobado una asignatura, junto con sus notas correspondientes.
- *pendientes*: devuelve una secuencia con las asignaturas que un determinado alumno no ha aprobado todavía.
- *junio*: modifica un *Tablon* eliminando de él la información relativa a las asignaturas aprobadas por los alumnos. De esta forma, en el *Tablon* sólo quedará información sobre las asignaturas pendientes de los alumnos.

Desarrolla en C++ una implementación de la clase *TTablon* basada en otros TADs conocidos, optimizando la complejidad temporal de las operaciones.

11. Nos han encargado desarrollar un sistema informático que se ocupe de la gestión de una tienda que vende libros por Internet. Para ello suponemos disponibles los módulos que implementan a las siguientes clases, todas ellas equipadas con operaciones de igualdad y orden:

- *TCliente*, que sirve para almacenar y gestionar la información sobre un cliente: su nombre, dirección, ….
- *TLibro*, que sirve para almacenar y gestionar la información sobre un libro: su título, autor, tema, ISBN, ….
- *TPais*, que sirve para representar los países a los que pertenecen los clientes. Suponemos que el *TCliente* está equipada con la observadora *país* que devuelve el país donde reside el cliente.
- *TTema*, que sirve para representar los temas de los libros. Suponemos que la clase T*Libro* está equipada con la observadora *tema* que devuelve el tema al que pertenece el libro.

Nosotros nos encargaremos de la implementación de la clase *TTienda*, cuya misión es gestionar la información sobre los libros disponibles y sobre los pedidos de los clientes. Esta clase ofrece las siguientes operaciones públicas:

- *Nuevo*: inicializa una *Tienda* vacía.
- *InsertaLibro*: añade a una *Tienda* un número determinado de ejemplares de un cierto libro. El libro podía estar ya disponible, con lo que esta operación sólo modificará el número de ejemplares, o puede que se trate de un libro nuevo.
- *Pedido*: esta operación añade a una *Tienda* la información de un pedido que ha realizado un cliente determinado. La información sobre el pedido está dada como una secuencia de libros. Aunque el envío de los libros no se realiza automáticamente al recibirse el pedido, los libros solicitados se consideran reservados, por lo que esta operación se deberá encargar de actualizar adecuadamente el número de ejemplares disponibles. También puede ocurrir que no haya existencias de alguno de los libros solicitados, en cuyo caso se retendrá la información sobre el pedido del libro o libros en cuestión, para así poder atenderlo en cuanto lleguen más ejemplares. Al insertarse nuevos ejemplares de un libro agotado para el que existen pedidos pendientes, dichos ejemplares se asignarán respetando el orden en el que se realizaron los pedidos.
- *envíoLibros*: esta operación consulta y modifica una *Tienda* para obtener la información sobre los pedidos pendientes de envío que tienen como destino a un país determinado, y darlos por realizados. Esta operación devuelve una secuencia donde cada elemento contiene los datos de un cliente y la secuencia de libros que ese cliente ha pedido (y tiene reservados). Nótese que entre los libros a enviar no se pueden incluir aquellos pedidos que están pendientes debido a la falta de ejemplares. Nótese asimismo que entre dos envíos sucesivos al mismo país, cada cliente puede haber realizado más de un pedido.
- *envíoCatálogo*: periódicamente, la librería edita catálogos con las novedades editoriales de un cierto tema. Esta operación sirve para consultar una *Tienda* y obtener la lista de clientes interesados en el tema en cuestión. Se considera que un cliente está interesado en un cierto tema si ha comprado más de *minLibros* libros de dicho tema.

Desarrolla en C++ una implementación de la clase *TTienda* basada en otros TADs conocidos, optimizando la complejidad temporal de las operaciones.

12. Nos han encargado desarrollar un sistema informático que se ocupe de la gestión de los pedidos en la pizzería la *Pizza Veloz*. Para ello suponemos disponibles los módulos que implementan a las siguientes clases:

- *TZona*, que sirve para representar una zona de la ciudad, y que se utiliza para organizar los envíos por zonas.
- *TDireccion*, que sirve para representar una dirección concreta. Suponemos que *TDireccion* ofrece una operación *zona* que devuelve un objeto de tipo *TZona* con la zona a la que pertenece dicha dirección.
- *TPizza*, que sirve para representar un tipo de pizza de entre las que se preparan en la *Pizza Veloz*.
- *THora*, que sirve para representar una hora del día.
- *TPedido*, que sirve para representar la información sobre un pedido y que está equipado con las operaciones:
  - *direccion*, que devuelve un objeto *TDireccion* con el destino del envío.
  - *cargamento*, que devuelve una secuencia de objetos *TPizza* con las pizzas a enviar.
  - *hora*, que devuelve la hora en la que se realizó el pedido.

Nosotros nos encargaremos de la implementación de la clase *TPizzeria*, cuya misión es gestionar la información sobre los pedidos pendientes. En concreto, esta clase estará equipada con las siguientes operaciones públicas:

- *Nuevo*: inicializa una *TPizzeria* vacía.
- *Inserta*: añade a una *TPizzeria* un nuevo pedido (*TPedido*).
- *comanda*: esta operación proporciona la información necesaria para que un motorista cargue su moto y atienda a un cierto número de pedidos. El resultado de esta operación será una secuencia de pares <dirección, secuencia de pizzas> que se obtendrá de la siguiente forma:
  - el pedido pendiente más antiguo pasa directamente a formar parte del resultado,
  - la comanda se completa con otros pedidos pendientes de la misma zona que el primero, teniendo en cuenta que: (1) se han de atender antes a los pedidos más antiguos, (2) no se debe superar el número de *maxPizzas* que caben en una moto, y (3) no se pueden fraccionar pedidos. De esta forma, se van considerando sucesivamente, por orden cronológico, los pedidos pendientes a la zona en cuestión e insertándolos en el resultado siempre que no se supere el número de *maxPizzas*. Este proceso se detiene cuando se ha conseguido exactamente una comanda con *maxPizzas* o cuando se han considerado todos los pedidos pendientes de envío a dicha zona. Se puede suponer, por último, que ningún pedido contiene más de *maxPizzas*.

Desarrolla en C++ una implementación de la clase *TPizzeria* basada en otros TADs conocidos, optimizando la complejidad temporal de las operaciones.

13. Nos han encargado desarrollar un sistema informático que se ocupe de la gestión de la reserva de películas en un pequeño video club. Para ello suponemos disponibles los módulos que implementan a las siguientes clases, todas ellas equipadas con operaciones de igualdad y orden:

- *TPelícula*, que sirve para representar una de las películas del video club.
- *TCliente*, que sirve para representar un socio del video club.

Nosotros nos encargaremos de la clase *TReservas*, cuya misión es gestionar la información sobre las películas reservadas. Esta clase ofrecerá las siguientes operaciones públicas:

- *nuevo*: inicializa un *TReservas* vacío.
- *reserva*: establece que un cliente ha reservado una cierta película.
- *reservasPendientes*: consulta cuántas reservas pendientes hay para una cierta película.
- *películasReservadas*: consulta qué películas tiene reservadas un cierto cliente.
- *primero*: consulta quién es el primer cliente que tiene reservada una cierta película.
- *alquila*: elimina la reserva más antigua de una cierta película.

Desarrolla en C++ una implementación de la clase *TReservas* basada en otros TADs conocidos, optimizando la complejidad temporal de las operaciones.

14. Nos han encargado desarrollar un sistema informático que se ocupe de las estadísticas sobre las medallas obtenidas en las Olimpiadas. Para ello suponemos disponibles los módulos que implementan a las siguientes clases:

- *TPais*, que sirve para representar un país.
- *TDeporte*, que sirve para representar un deporte olímpico.
- *TPrueba*, que sirve para representar una prueba concreta de un cierto deporte y que está equipada con la operación *deporte* que devuelve un objeto *TDeporte* con el deporte al que pertenece dicha prueba.
- *TAtleta*, que sirve para representar a un atleta de los que participan en las Olimpiadas y que está equipada con la operación *pais* que devuelve un objeto *TPais* con el país al que pertenece el atleta.

Nosotros nos encargaremos de la clase *TMedallero*, cuya misión es gestionar la información sobre el número de medallas obtenido por los países participantes en las Olimpiadas, y de la clase *TMedallas* que representa el número de medallas obtenido por un país concreto (cuántas de oro, cuántas de plata y cuántas de bronce). La clase *TMedallero* estará equipada con las siguientes operaciones públicas:

- *Nuevo*: inicializa un *TMedallero* vacío.
- *clasificacion*: añade a un *TMedallero* la información sobre la clasificación de una prueba, dada como un objeto *TPrueba* y una secuencia de objetos *TAtleta* ordenada por el puesto obtenido en dicha prueba.
- *medalleroDeporte*: para un deporte dado devuelve una secuencia de pares <*TPais, TMedallas*> con los países que han obtenido alguna medalla en dicho deporte, ordenada por el número de medallas.
- *medallero*: devuelve una secuencia de pares <*TPais, TMedallas*> con los países que han obtenido alguna medalla, ordenada por el número de medallas.
  Para obtener el orden en el medallero, se consideran primero el número de medallas de oro, si éstas coinciden, se comparan entonces las de plata, y si también coinciden éstas, las de bronce. Si coinciden el número de medallas de oro, plata y bronce, entonces no importa el orden.

Desarrolla en C++ una implementación de las clases *TMedallas* y *TMedallero* basada en otros TADs conocidos, optimizando la complejidad temporal de las operaciones.

15. Nos han encargado desarrollar un revisor de textos. Nuestro sistema gestionará una colección con palabras del castellano y posibles formas de escribirlas mal, suponiendo que una palabra mal escrita se corresponde con una única palabra correcta. El proceso de corrección consistirá en, dado un texto, revisarlo palabra por palabra, de forma que si la palabra es correcta entonces se deja como está, si no es correcta pero sabemos a qué palabra correcta se refiere la sustituimos, y si el sistema no la conoce se deja tal cual.

La clase *TCorrector* que implementa el sistema ofrecerá las siguientes operaciones públicas:

- *nuevo*: inicializa un *TCorrector* vacío.
- *insertaPalabra*: añade una nueva palabra correcta.
- *insertaError*: añade una nueva forma de escribir mal una cierta palabra, es decir, asocia una nueva palabra incorrecta con una correcta.
- *revisa*: dado un texto representado por una secuencia de palabras, le aplica el proceso de corrección antes descrito, devolviendo el texto corregido junto con una secuencia que contenga las palabras que el sistema desconoce –no sabe si son correctas o incorrectas–.
- *diccionario*: devuelve una secuencia ordenada alfabéticamente que contenga cada palabra correcta y las formas posibles de escribirla mal.

Desarrolla en C++ una implementación de la clase *TCorrector* basada en otros TADs conocidos, optimizando la complejidad temporal de las operaciones. En la implementación está prohibido el acceso a los tipos representantes de los TADs que se usen.

Para cada operación, indica la complejidad temporal que has obtenido.

16. Nos han encargado desarrollar un sistema informático que gestione un foro de mensajes. En el foro hay dos tipos de mensajes: los que inician una "línea de discusión", es decir, aquellos mensajes nuevos que no responden a un mensaje anterior; y los que responden a un mensaje previo (ya sea una respuesta a otra respuesta o una respuesta a un mensaje inicial).

La clase *TForo* que implementa el sistema ofrecerá las siguientes operaciones públicas:

- una constructora sin parámetros que inicialice un *TForo* vacío.
- *insertaMensaje*: añade un mensaje que inicia una línea de discusión.
- *insertaRespuesta*: inserta un mensaje *a* como respuesta directa a un mensaje previo *b*.
- *iniciales*: obtiene los mensajes almacenados en el foro que han iniciado una línea de discusión.
- *respuestas*: obtiene las respuestas directas a un mensaje dado.
- *mensajes*: obtiene todos los mensajes del foro.

Escribe la definición de la clase *TMensaje* (sólo la interfaz) y desarrolla en C++ una implementación de la clase *TForo* basada en otros TADs conocidos, optimizando la complejidad temporal de las operaciones. Si en la clase *TMensaje* decides incluir alguna operación no trivial, entonces has de implementar también dicha operación.

Para cada operación, indica la complejidad temporal que has obtenido.

17. Nos han encargado desarrollar un sistema informático que gestione un sistema de indexación de documentos mediante palabras clave relacionadas. Cuando se inserta un documento, se le asocia una palabra clave que luego permitirá recuperarlo (decimos entonces que el documento está *indexado* por esa palabra clave). La misma palabra clave puede tener varios documentos asociados.

Por otra parte, las palabras clave están relacionadas entre sí por relaciones de dependencia, de forma que si la palabra clave $c1$ depende de la palabra clave $c2$, entonces cada vez que se consulte por $c1$ se deberán recuperar también los documentos asociados con $c2$. La relación de dependencia es transitiva (si $c1$ depende de $c2$ y $c2$ depende de $c3$ entonces $c1$ depende de $c3$) y antisimétrica (si $c1$ depende de $c2$ entonces $c2$ no puede depender de $c1$).

La clase *TGestor* que implementa el sistema ofrecerá las siguientes operaciones públicas:

- una constructora sin parámetros que inicialice un *TGestor* vacío.
- *insertaClaves*: añade una relación de dependencia entre dos claves dadas. No es necesario que las claves hayan sido insertadas previamente en el sistema. Se valorará que esta operación compruebe que no se producen ciclos en la relación de dependencia entre claves.
- *insertaDocumento*: añade un documento con una palabra clave asociada. No es necesario que la clave haya sido insertada previamente.
- *consulta*: dada una cierta palabra clave, recupera sus documentos asociados junto con los documentos asociados con las claves de las que depende directa o indirectamente, si es que existen.

Escribe la definición de la clase *TDocumento* (sólo la interfaz) y, suponiendo que las palabras clave se representan como datos de tipo *string*, desarrolla en C++ una implementación de la clase *TGestor* basada en otros TADs conocidos, optimizando la complejidad temporal de las operaciones. Si en la clase *TDocumento* decides incluir alguna operación no trivial, entonces has de implementar también dicha operación.

Para cada operación, indica la complejidad temporal que has obtenido.