0. Teoría general

1. Genericidad

Es el que define un tipo abstracto de datos con un comportamiento común, pero que se diferencia en detalles que no afectan a dicho comportamiento,

La generalización se emplea especialmente en la creación de TAD contenedores como son las pilas, colas y listas, diseñados para almacenar una colección de elemento de un mismo tipo.

2. Tipos de medida de un algoritmo:

Medida Directa: El valor de la magnitud a medir se obtiene directamente, se observa directamente el valor numérico en un instrumento de medida adecuado

Medida Indirecta: Se obtiene mediante la medición de otras magnitudes relacionadas con ella, por lo que debe manipularse matemáticamente una o varias medidas directas.

Medida adaptativa: Dentro de la medida indirecta, con la medida relativa se mide un número finito de veces hasta que el error dado es el mínimo posible.

3. Principio de invarianza:

Dos programas correspondientes a un mismo algoritmo sólo difieren en eficiencia temporal en un factor constante, por tanto aun mejorando la máquina donde corre el algoritmo o el programa, solo cambiará en un factor constante.

1. Preguntas generales

1. ¿Qué es un TAD?

Re1: Colección de datos con las mismas características y especificación de sus operaciones y propiedades.
Re2: Un tipo abstracto de dato es una colección de valores, junto con unas operaciones sobre ellos, definidos mediante una especificación que es independiente de cualquier implementación. (definición del libro “Abstracción y estructuras de datos en c++”).

Re3:Un TAD es un conjunto de valores y operaciones que se definen mediante una especificación independiente de cualquier representación. El calificativo de abstracto expresa la cualidad de independencia de la representación.

2. ¿Qué importancia tiene la especificación en la creación y utilización de un TAD?

El uso y la creación de un TAD se basa en su especificación, siendo independiente de la representación subyacente. La especificación de un TAD informa de qué hace sin entrar en detalles internos de su implementación (cómo lo hace), es decir, la especificación comprende toda la información que necesita conocer el usuario del TAD: el Dominio (conjunto de datos del TAD, objetos pertenecientes al TAD) y la especificación de operaciones (Especificación sintáctica: cómo usar las operaciones, Especificación semántica: qué hace y qué propiedades tiene cada una), esta es la única parte que conoce el usuario del mismo y contiene el nombre del tipo y la especificación de las operaciones. Es la parte más importante del TAD.

3. ¿Qué es la especificación sintáctica?

Es la que indica cómo usar las operaciones, es decir, la forma en que se ha de escribir cada una, dando su nombre junto al orden y tipo de operandos y resultados.

4. ¿Qué es la especificación semántica?

Es la que expresa el significado de las operaciones, o sea, qué hace y qué propiedades tiene cada una.

5. ¿Qué es la implementación de un TAD?

La implementación de un TAD es la parte conocida únicamente por el diseñador y consiste en el diseño de una estructura de datos que represente los elementos del dominio del TAD y el código que implementa los algoritmos de tratamiento de dicha estructura, descritos en la fase de especificación.

6. Principio fundamental de los TAD: independencia de la representación.

Como un TAD se basa en su especificación y, esta es independiente de cualquier representación del mismo que se pueda diseñar, implica que un mismo TAD se pueda implementar con distintas representaciones. Los requisitos del problema o aplicación determinarán la representación más adecuada a utilizar.

7. ¿Existe alguna relación entre la ocultación de información y la independencia de la representación?

Re1: Mediante la ocultación de información podemos separar la interfaz de un TAD (características y comportamientos visibles desde el exterior) de los detalles internos de la implementación de la representación que se use.

Re2: En programación, el concepto de abstracción de un problema se consigue mediante la abstracción operacional y la abstracción de datos. Ambas, implican ocultación de información, ya que al usuario no le interesa saber cómo se están implementadas las operaciones ni cómo se almacenan los datos. La abstracción de datos se consigue mediante la utilización de tipos abstractos de datos (TADs), los cuáles, siguen el principio de independencia de la representación, que asegura al usuario que las operaciones y el dominio del TAD descritos en la especificación, van a comportarse tal y como se especifica independientemente de la representación que se use.
Re3: Sí, ya que gracias a la ocultación de información conseguimos que la especificación de un TAD sea independiente de su implementación.

8. Ante la posibilidad de que no se verifiquen las precondiciones de una operación de un TAD, ¿qué decisiones de diseño puede implementar la operación?

Las precondiciones de una operación indican las condiciones que deben cumplirse antes de la utilización de dicha operación para su correcto funcionamiento, y el diseñador está obligado a implementar la operación de modo que asegure el comportamiento asegurado. En caso de no cumplirse la precondición no se asegura que el comportamiento sea el esperado. En este caso, el diseñador de dicha operación puede tomar la decisión de notificar con un mensaje de error al usuario indicando esto mismo.

9. ¿Qué sucede si después de la ejecución de una determinada operación, no se cumplen las precondiciones de la misma?

En caso de que no se cumplan las precondiciones no se puede asegurar el correcto funcionamiento de la operación. Puede darse el caso de que se detenga la ejecución del programa, o puede acabar la ejecución de la operación, pudiendo dar un resultado erróneo o indeterminado.

10. ¿Cómo se realiza la ocultación en C?

Mediante la abstracción de las implementaciones de las operaciones.

Debería ser también la ocultación de los tipos de datos, pero esto en C no es posible ya que no podemos ocultar el archivo de la cabecera .h.

Las implementaciones se introducen en un archivo .c oculto (que sería el archivo .o de ese fichero) y se le proporcionaría al usuario del TAD especificándole las operaciones que puede utilizar.

11.¿Qué ventajas tiene realizar la especificación de un TAD después de hacer la implementación?
No solo no tiene ventajas, sino que además no es posible. Porque el principio de independencia nos dice que independientemente de la representación que se use, la especificación del TAD es siempre la misma.

2. Pila

1. ¿Qué es una pila?

Una pila es una secuencia de elementos en la que todas las operaciones se realizan por un extremo de esta. Dicho extremo recibe el nombre de tope, cima, cabeza. . . En una pila el último elemento añadido es el primero en salir de ella, por lo que también se les conoce como estructuras LIFO: Last Input First Output.

2. ¿Cuándo utilizamos una representación estática del TAD Pila? Ventajas e inconvenientes.

Esta representación se utilizará cuando sepamos en **tiempo de compilación** el número máximo de elementos que se almacenará en la pila.

Ventajas: Ocupa menos espacio que la representación del TAD Pila por celdas enlazadas, ya que no hay que almacenar un puntero por cada elemento.

Inconvenientes: Exige fijar el tamaño de la pila en tiempo compilación, por lo que el tamaño será constante y todas las pilas que creemos tendrán el mismo tamaño. Además, del posible desaprovechamiento del espacio si no se llega a ocupar todo el espacio reservado.

3 ¿Cuándo utilizaremos una representación pseudoestática del TAD Pila? Ventajas e inconvenientes.

Usaremos la representación pseudoestática cuando queramos dejar al usuario que determine el número máximo de elementos que contenga la pila.

Ventajas: Nos permite fijar distintos tamaños en **tiempo de ejecución** (este tamaño será fijo durante todo el tiempo de vida de la pila), por lo que el tamaño inicial no determina el tamaño de todas. Ocupa menos espacio que la representación mediante celdas enlazadas ya que no necesita almacenar un puntero en cada elemento de la pila.

Inconvenientes: Tamaño constante durante la vida de cada pila, con el posible desaprovechamiento del espacio si no se llega a ocupar todo el espacio reservado.

4. ¿Cuándo utilizaremos una representación dinámica del TAD Pila? Ventajas e inconvenientes.

Utilizaremos la representación dinámica del TAD Pila cuando no sepamos cuánto espacio necesitamos como máximo o no tengamos que controlar el número máximo de elementos que tenga la pila.

Ventajas: El tamaño de las pilas es variable, con lo que podemos incrementar o decrementar cuando queramos en **tiempo de ejecución**. Siempre ocupará el espacio justo para almacenar los elementos que contenga la pila en cada momento.
Inconvenientes: Alto consumo de memoria debido a la utilización de un puntero por cada elemento.

5. ¿Tiene algún sentido representar una pila con una lista doblemente enlazada?
No, primero porque estaríamos ocupando más espacio del que necesitamos al tener otro puntero “anterior” por cada elemento. Además, según la especificación del TAD Pila, las operaciones se realizan solo por uno de sus extremos, mientras que con las listas tenemos acceso directo a cualquier posición de esta.

6. ¿Cuándo utilizaremos la representación circular del TAD Pila?
Nunca, ya que no existe esta representación.

7. ¿Cómo se podría eliminar un elemento cualquiera de una cola/pila?
No se podría hacer, ya que la especificación de pila y cola no permite hacerlo, ya que todas las operaciones de la Pila se realizan por uno de los extremos, y la operación eliminar de la Cola también se realiza por un solo extremo, el frente. En la pila las eliminaciones se hacen por lo que llamamos tope. Una forma de hacerlo sin saltarnos la especificación sería ayudarnos de una pila/cola auxiliar en la cual volcamos la que tenemos, hacer pop en el elemento que queramos eliminar y después hacer otro volcado.

8. ¿Por qué en las implementaciones mediante celdas enlazadas, tanto de pilas como de colas, no existe un método llena?

Porque, tanto la pila como la cola nunca van a estar llenas, ya que se inicializan dinámicamente, es decir el tamaño no está definido y, por tanto, el tamaño de la estructura variará en tiempo de ejecución tanto como el usuario quiera.

9. ¿En qué programas utilizarías el TAD Pila?

Utilizaremos el TAD Pila en aquellos casos en los que el usuario desee extraer los elementos en el ***orden inverso*** en que se insertaron y viceversa. ¿También si se desea que todas las operaciones del TAD tengan orden constante? Algunos programas comunes en los que se usa el TAD Pila son por ejemplo líneas de texto.

10. ¿Porqué la representación estática y pseudoestática del TAD Pila, el registro que representa a la pila se pasa por referencia sin ser necesario?

No es necesario pasarlo por referencia. Sin embargo, en virtud del principio de independencia de la representación, los parámetros del tipo pila tienen que ser transferido por referencia en las diferentes representaciones.

11. ¿Por qué en la representación dinámica del TAD Pila se representa la pila como un puntero a las estructuras de datos diseñadas?

La razón de introducir un puntero es hacer que el paso de parámetros a las operaciones del TAD sea por referencia, de modo que los cambios en los parámetros formales sean reflejados en los parámetros reales, preservando así el principio de independencia de la representación.

3. Colas

1. ¿Qué es una cola?

Una cola es una secuencia de elementos en la que las operaciones se realizan por los extremos: Las eliminaciones se realizan por el extremo llamado inicio, frente o principio de la cola. Los nuevos elementos son añadidos por el otro extremo, llamado fondo o final de la cola. En una cola el primer elemento añadido es el primero en salir de ella, por lo que también se les conoce como estructuras FIFO: First Input First Output.

2. Ventajas e inconvenientes de una representación vectorial del TAD Cola.

-Ventajas: Ocupa un espacio fijo de memoria, el cual está representado por una estructura que contiene un vector que almacena todos los elementos, un entero que almacena el tamaño máximo del vector y otro entero que almacena la posición del último elemento de la cola. También, ocupa menos espacio que la representación mediante celdas enlazadas, ya que no hace falta almacenar un puntero por cada elemento de la cola.

Inconvenientes: El número máximo de elementos es constante en todo el tiempo de vida de la cola, al ser una representación pseudoestática. Y además, en la operación pop(), se habría de mover todos los elementos de la cola una posición anterior. También, existe la posibilidad de desaprovechar el espacio almacenado si no se rellenan todas las posiciones de la cola.

3. Ventajas e inconvenientes de la representación circular del TAD Cola.

Ventajas: Esta representación soluciona el inconveniente de la representación vectorial de tener que desplazar los elementos cada vez que utiliza la operación pop().

Inconvenientes: Es distinguir entre una cola llena y una cola vacía, ya que la posición de los índices de inicio y fin sería la misma en ambos casos. Las soluciones propuestas serían añadir un indicador (booleano) de si está o no vacía la cola junto con un contador de elementos o añadir una posición más al vector, la cual siempre estaría vacía: Si inicio sigue a fin, la cola está vacía, si entre inicio y fin hay una

posición vacía, cola llena y si inicio = fin solo hay un elemento. Además, el número máximo de elementos es constante en todo el tiempo de vida de la cola, al ser una representación pseudoestática.

4. Eres un diseñador del TAD Cola y un usuario te solicita una operación de acceso al n-ésimo elemento de la misma. ¿Incluirías esta operación en el TAD?

No, porque estaríamos incumpliendo la especificación del TAD Cola, que no incluye esta operación, y el TAD no se puede cambiar. Para incluir esa operación crearíamos otro TAD y en la especificación de este TAD incluiríamos dicha operación.

5. ¿Por qué en la implementación vectorial circular de las colas, existe una posición que siempre está vacía en el vector?

Existe una posición que siempre deba estar vacía para que se pueda distinguir cuando está la cola vacía y cuando está llena. Estará vacía cuando inicio siga a fin, y llena, cuando entre ambas posiciones quede una libre, la que hemos sacrificado.

6. La representación de una cola mediante estructuras enlazadas, ¿por qué se representa usando dos punteros a cada extremo de la misma?(Ventajas e Inconvenientes)

La representación de una cola mediante estructuras enlazadas podría realizarse utilizando un solo puntero a uno de los extremos de la misma, o mediante dos punteros, uno a cada extremo, como en este caso. Una posible representación, utilizando tan solo un puntero, sería apuntar al fin de la misma y enlazar el último elemento con el primero, de forma que se accedería al fin de la cola en coste O(1), y al inicio en coste O(2). Otra posibilidad, acorde con los accesos al TAD cola, es representarla mediante dos punteros que apunten al inicio y fin de la cola. Su ventaja es que el acceso a ambos extremos de la cola se consigue en coste O(1), mejorando a la representación con un único puntero. Su inconveniente es que requiere dos punteros en vez de uno para representar la misma (más memoria).

7. ¿Para qué se utiliza el nodo cabecera en el TAD Cola, en su implementación con celdas enlazadas?

El nodo cabecera no se utiliza en el TAD cola, en ninguna de sus representaciones, se utiliza en las listas para cumplir con la especificación de poder representar cada posición de un nodo como el puntero al nodo anterior.

8. ¿Con la implementación de colas mediante un vector circular de tamaño n cuántos elementos pueden almacenarse?

Si dejamos una posición vacía para diferenciar cuando está la cola vacía y cuando llena, podríamos tener hasta n-1 elementos. En cambio, si para controlar el estado de la cola, es decir, si esta vacía o llena usamos un indicador (booleano) y un contador de elementos podríamos tener n elementos en nuestra cola.

9. Con la representación de colas mediante una estructura enlazada, con puntero al final y circular. ¿Cuántos elementos pueden almacenarse?

Podemos almacenar todos los que queramos, ya que esta representación se hace mediante memoria dinámica, por tanto, no hay un límite de elementos a almacenar. El límite lo marca la capacidad de memoria del dispositivo donde se ejecute el programa que haga uso del TAD Cola con dicha representación.

10. Comenta la siguiente afirmación: El TAD Cola Circular es un TAD representado mediante un vector circular en el que el número de elementos a insertar como máximo es n-1.

El TAD Cola Circular no existe, pero sí existe una representación circular del TAD Cola, en el que se pueden almacenar n-1 elementos, debido al uso de una posición de esta como indicador de cola llena/vacía.

11. ¿En qué programas utilizarías el TAD Cola?

Utilizaremos el TAD Cola en aquellos casos en los que el usuario desee extraer los elementos en el mismo orden en que se insertaron. También para programas que representen líneas de espera. También si se desea que todas las operaciones del TAD tengan orden constante

12.Define una estructura de datos basada en el vector circular para almacenar una bicola de valores de tipo T y escribe el código de las operaciones: void BicolaPopFin(Bicola B) y void BicolaPushInicio(T x, Bicola B).

```
class bicola {
        public:
                bicola(size_t tmax);
                void BicolaPopFin();
                void BicolaPushInicio(const T& x);

        private:
                T *elemento;
                int tmax,lmax;
                int inicio,fin;
 };

Bicola(size_t tmax){
elemento(new T[tmax+1]);
lmax = tmax+1;
inicio = 0;
fin = tmax;
}

void BicolaPopFin(){
assert(!vacia());
fin = fin-1 % Lmax;
 }
void BicolaPushInicio(const T& x){
assert(!llena());
inicio = inicio-1 % Lmax;
elemento[inicio] = x ;
}
```

4. Listas

1. ¿Qué es una lista?

R1: Secuencia de elementos del mismo tipo, (a1, a2, . . . , an), cuya longitud, n > 0, es el número de elementos que contiene. Si n = 0, es decir, si la lista no tiene elementos, se denomina lista vacía.
R2: Una lista es una secuencia de elementos de un tipo determinado. Se representa de la forma L = (a1, a2, ..., an), donde n ≥ 0 y longitud = n. Si n = 0 es una lista vacía.

Posición: Lugar que ocupa un elemento en la lista. Los elementos están ordenados de forma lineal según las posiciones que ocupan. Todos los elementos salvo el primero tienen un único predecesor, y todos los elementos salvo el último tienen un único sucesor.
Posición Fin: Posición siguiente a la del último elemento. Esta posición no está ocupada nunca por ningún elemento.

2. ¿En una representación vectorial de una lista, como representamos la posición fin?
La posición fin corresponde al número que contiene el total de elementos que hay en la lista, es decir, corresponde a n.

3.Ventajas e inconvenientes de la implementación vectorial del TAD lista

Ventajas: Las posiciones podrían representarse por enteros y, además, se pueden permitir el acceso directo a un elemento en concreto por dicho entero.

Inconvenientes: Si la lista es demasiado pequeña, puede ser que se desaproveche espacio para albergar al puntero de cada nodo que a la propia lista.

4. Ventajas e inconvenientes de la implementación dinámica del TAD Lista

Ventajas: No se desaprovecha memoria con el uso de punteros, ya que se usan el espacio necesario.

Inconvenientes: Si la lista es demasiado pequeña, puede ser que se “desaproveche” más espacio para albergar al puntero de cada nodo que a la propia lista.

5.¿Por qué surge el nodo cabecera en el TAD Lista?
Porque si no existiese el nodo cabecera, nos estaríamos saltando el principio de la independencia de la representación, así pues, utilizando el nodo cabecera estaremos cumpliendo la especificación del TAD en las funciones eliminar e insertar para que todos los elementos tengan predecesor y así ninguno tenga que tratarse de forma distinta. Con este nodo las funciones eliminar e insertar pasan a ser de orden 1 ya que con la cabecera no es necesario recorrer la lista en busca de la posición anterior de las anteriores funciones.

6. ¿Cuando se utiliza el TAD Lista Circular?
Cuando no hay definido un primer y un último elemento de la lista, es decir, cuando no se sabe cuáles son los extremos de la lista, por lo que se puede acceder a todos los nodos a partir de uno cualquiera. También, hay que tener en cuenta que es innecesario el nodo cabecera y la operación fin().

7. Diferencia entre TAD Lista Circular y TAD Lista
La diferencia principal entre estos dos TADS es que , en la lista circular, a diferencia de la Lista, todos los elementos tienen estrictamente un sucesor y un predecesor. Una lista circular no posee posición primera() ni posición fin(), solo posee una posición llamada inipos() que devuelve una posición para tomarla como referencia. También cabe decir que el TAD Lista Circular, a diferencia del TAD Lista, no posee cabecera ya que es innecesaria.

8.¿Para qué se utiliza una implementación mediante estructura enlazada doble?
Se utiliza para reducir el orden de las operaciones anterior y fin, de orden n a 1, ya que, en esta implementación, existe, además del puntero a la posición siguiente un puntero a la anterior. La única operación que sería de orden n es la operación buscar que no se puede mejorar.

9.¿Cuándo es conveniente utilizar una implementación mediante una estructura enlazada doble?
Es conveniente utilizar esta implementación cuando se vayan a realizar operaciones como anterior y fin, para evitar el mayor coste de éstas.

10. En la lista circular, ¿es posible implementar alguna operación de orden logarítmico?
No, todas las operaciones son de orden 1 o de orden n , ya que se referencia al elemento al cual apunta el puntero o al contiguo, o para las operaciones del orden n, recorre toda la lista circular hasta hallar el elemento que se desea.

11.¿Qué condición tiene que cumplir una lista para que la búsqueda sea logarítmica?¿Y de orden cuadrático?

Una lista no podrá tener orden logarítmico en la búsqueda. Y una búsqueda de orden cuadrática no sería factible ya que el orden normal de las búsquedas es n y con este orden lo empeoraríamos.

12. ¿Por qué la especificación del TAD Lista en la operación anterior, las precondiciones son L = (a1...an) 2 menor o igual que p, menor igual que n+1?

El hecho de que p deba ser mayor o igual que 2 y menor o igual que n + 1, viene dado porque la primera posición no tiene anterior y la posición fin, aunque no tenga elementos, si posee una posición anterior.

13.¿Qué pasaría si insertaras o eliminaras en la primera posición en una implementación de la lista mediante estructura enlazada sin cabecera?

Se perdería la asignación de la posición al no tener el primer elemento un predecesor definido, hecho que con la cabecera no pasaría.

14. Representando el TAD Lista mediante un vector, ¿Es posible evitar el coste n para inserciones y borrados en los extremos?

Si, implementamos el TAD Lista mediante un vector circular.

15. ¿En qué programas usarías el TAD Lista?

Lo usaremos en aquellos casos en los que se desee insertar y eliminar elementos en posiciones concretas de la estructura.

16.¿Con qué problema fundamental nos encontramos si suprimimos L en los argumentos de las operaciones del TAD lista?

Nos estamos saltando el principio de independencia de la representación del TAD, en las operaciones: Insertar(), Eliminar(), Anterior() y Siguiente().
Si suprimimos L en los argumentos, las especificación y utilización del TAD dejan de ser independientes de la implementación.

17.¿Por que en la especificación del TAD lista, en la operación siguiente, las precondiciones son L=(a1,a2,...,an) 1 <= p <= n?

El hecho de que p deba ser mayor o igual que 1 y menor o igual que n viene dado porque el último elemento no tiene siguiente (el siguiente es la posición fin).

18.¿Por qué en la especificación del TAD Lista Circular, en las operaciones anterior y siguiente, las precondiciones son L=(a1,a2,...,an) 1 <= p <= n ?

Porque todas y cada una de las posiciones de la lista circular poseen una posición anterior y una posición siguiente.

19.¿Por qué se pasa el parámetro posición por referencia en una lista en las operaciones insertar y eliminar en la implementación mediante estructuras enlazadas?

Al modificar el 1er elemento será necesario cambiar ese puntero inicial para que apunte al nuevo primer elemento, así que ese puntero hay que pasarlo por referencia.