Tipo de dato abstracto

Un **tipo de dato abstracto** (**TDA**) o **tipo abstracto de datos** (**TAD**) es un modelo matemático compuesto por una colección de operaciones definidas sobre un conjunto de datos para el modelo.

Índice

Introducción

En el mundo de la programación existen diversos lenguajes de programación que se han ido creando con el paso del tiempo y que se han perfeccionado debido a las necesidades de los programadores de la época a la que pertenecen. Los primeros lenguajes de programación eran de tipo lineal, ya que un programa se recorría desde un punto marcado como Inicio hasta llegar a un punto Fin. Con el tiempo se fueron creando nuevos lenguajes y en nuestros días los más utilizados son los llamados “orientados a objetos”.

Los lenguajes orientados a objetos (LOO) tienen la característica de que no son lenguajes lineales, sino que se forman de diversas funciones, las cuales son llamadas en el orden en que el programa mismo las pide o el usuario determina. Para entender mejor cómo funcionan los lenguajes orientados a objetos, vamos a introducir un concepto fundamental en las Estructuras de Datos denominado Abstracción de Datos y que es parte importante de estos Lenguajes y de la manera en que funciona la mayoría del software comercial de nuestros días.

Historia

El concepto de tipo de dato abstracto (TDA, Abstract Data Type), fue propuesto por primera vez hacia 1974 por John Guttag y otros, pero no fue hasta 1975 que por primera vez Barbara Liskov lo propuso para el lenguaje CLU.

El lenguaje Turbo Pascal fue determinante para la común aceptación de los TDA con la introducción de las Units, si bien estas no cumplen con las características básicas de un tipo de dato abstracto como por ejemplo la encapsulación de los datos. El lenguaje de programación Ada pudo implementar exitosamente los TDA con sus *packages*. Vale recordar que estos dos últimos lenguajes soportan formalmente la Programación modular.

Definición

Con mucha frecuencia se utilizan los términos *TDA* y *Abstracción de Datos* de manera equivalente, y esto es debido a la similitud e interdependencia de ambos. Sin embargo, es importante definir por separado los dos conceptos.

Como ya se mencionó, los Lenguajes de Programación Orientados a Objetos son lenguajes formados por diferentes métodos o funciones y que son llamados en el orden en que el programa lo requiere, o el usuario lo desea. La abstracción de datos consiste en ocultar las características de un objeto y obviarlas, de manera que solamente utilizamos el nombre del objeto en nuestro programa. Esto es similar a una situación de la vida cotidiana. Cuando yo digo la palabra “perro”, usted no necesita que yo le diga lo que hace el perro. Usted ya sabe la forma que tiene un perro y también sabe que los perros ladran. De manera que yo abstraigo todas las características de todos los perros en un solo término, al cual llamo “perro”. A esto se le llama ‘abstracción’ y es un concepto muy útil en la programación, ya que un usuario no necesita mencionar todas las características y funciones de un objeto cada vez que este se utiliza, sino que son declaradas por separado en el programa y simplemente se utiliza el término abstracto (“perro”) para mencionarlo.

En el ejemplo anterior, “perro” es un Tipo de Dato Abstracto y todo el proceso de definirlo, implementarlo y mencionarlo es a lo que llamamos Abstracción de Datos.

Vamos a poner un ejemplo real de la programación. Supongamos que en algún Lenguaje de Programación Orientado a Objetos un pequeño programa saca el área de un rectángulo de las dimensiones que un usuario decida. Pensemos también que el usuario probablemente quiera saber el área de varios rectángulos. Sería muy tedioso para el programador definir la multiplicación de ‘base’ por ‘altura’ varias veces en el programa, además que limitaría al usuario a sacar un número determinado de áreas. Por ello, el programador puede crear una función denominada ‘Área’, la cual va a ser llamada, con los parámetros correspondientes, el número de veces que sean necesitadas por el usuario y así el programador se evita mucho trabajo, el programa resulta más rápido, más eficiente y de menor longitud. Para lograr esto, se crea el método Área de una manera separada de la interfaz gráfica presentada al usuario y se estipula ahí la operación a realizar, devolviendo el valor de la multiplicación. En el método principal solamente se llama a la función Área y el programa hace el resto.

Al hecho de guardar todas las características y habilidades de un objeto por separado se le llama Encapsulamiento y es también un concepto importante para entender la estructuración de datos. Es frecuente que el Encapsulamiento sea usado como un sinónimo del Ocultación de información, aunque algunos creen que no es así [1]

Separación de la interfaz e implementación

Cuando se usa en un programa de computación, un TDA es representado por su interfaz, la cual sirve como cubierta a la correspondiente implementación. La idea es que los usuarios de un TDA tengan que preocuparse solo por la interfaz, pero no por la implementación, ya que esta puede ir cambiando con el tiempo y, si no existiera encapsulación, afectar a los programas que usan el dato. Esto se basa en el concepto de Ocultación de información, una protección para el programa de decisiones de diseño que son objeto de cambio.

La solidez de un TDA reposa en la idea de que la implementación está escondida al usuario. Solo la interfaz es pública. Esto significa que el TDA puede ser implementado de diferentes formas, pero mientras se mantenga consistente con la interfaz, los programas que lo usan no se ven afectados.

Hay una diferencia, aunque a veces sutil, entre el Tipo de Dato Abstracto y la Estructura de Datos usada en su implementación. Por ejemplo, un TDA de una lista puede ser implementado mediante un Arreglo o una Lista Enlazada o hasta un Árbol binario de búsqueda. Una lista es un Tipo de Dato Abstracto con operaciones bien definidas (agregar elemento, agregar al final, agregar al principio, recuperar, eliminar, etc) mientras una lista enlazada es una estructura de datos basada en punteros o referencias (dependiendo del lenguaje) que puede ser usada para crear una representación de una Lista. La Lista Enlazada es comúnmente usada para representar una TDA Lista, y a veces, hasta confundida. Un ejemplo es la clase Linked List de Java, la cual ofrece una gran cantidad de métodos que no corresponden a una Lista Enlazada "pura", sino a un fuerte TDA.

De forma similar, un TDA Árbol binario de búsqueda puede ser representado de muchas maneras: Árbol binario, Árbol AVL, Árbol rojo-negro, Arreglo, etc. A pesar de la implementación un Árbol binario siempre tiene las mismas operaciones (insertar, eliminar, encontrar, etc.)

Separar la interfaz de la implementación no siempre significa que el usuario ignora totalmente la implementación de la rutina, pero lo suficiente para no depender de ningún aspecto de la implementación. Por ejemplo, un TDA puede ser creado usando un script o cualquiera que pueda ser descompilado (como C).

Caracterización

Un TDA está caracterizado por un conjunto de operaciones (funciones) al cual se denomina usualmente como *interfaz pública* y representa el comportamiento del TDA; mientras que la *implementación* como la parte privada del TDA está oculta al programa cliente que lo usa. Todos los lenguajes de alto nivel tienen predefinidos TDA; que son los tipos denominados simples y las estructuras predefinidas, y estos tienen sus interfaces públicas que incluyen las operaciones como la +, -, *, etc.

No se necesita conocer como actúan tales operadores sobre la representación interna de los tipos definidos, que además, suele ser una implementación bastante dependiente de la máquina sobre la que trabaje el compilador. Lo interesante es que los lenguajes actuales nos van a permitir ampliar los TDA predefinidos con otros que serán definidos por el propio programador para adecuar así los tipos de datos a las necesidades de los programas.

Los TDA que nos van a interesar de ahora en adelante son aquellos que reflejen cierto comportamiento organizando cierta variedad de datos estructuradamente. A esta forma estructurada de almacenar los datos será a la que nos refiramos para caracterizar cada TDA.

Los TDA que tienen informaciones simples pero dependientes de un comportamiento estructural serán llamados políticos y aquellos TDA simples, como son los tipos predefinidos donde la información no es relacionada mediante ninguna estructura y no admiten más que un valor en cada momento será denominados TDA monolíticos.

Nótese que cuando hablemos de un TDA no haremos ninguna alusión al tipo de los elementos sino tan solo a la forma en que están dispuestos estos elementos. Solo nos interesa la estructura que soporta la información y sus operaciones. Para determinar el comportamiento estructural basta con observar la

conducta que seguirán los datos.

Caractericemos entonces los TDA. Un TDA tendrá una parte que será invisible al usuario la cual hay que proteger y que se puede decir que es irrelevante para el uso del usuario y está constituida tanto por la maquinaria algorítmica que implemente la semántica de las operaciones como por los datos que sirvan de enlace entre los elementos del TDA, es decir, información interna necesaria para la implementación que se esté haciendo para ese comportamiento del TDA. Resumiendo podemos decir, que tanto la implementación de las operaciones como los elementos internos del TDA serán privados al acceso externo y ocultos a cualquier otro nivel.

Un TDA representa una abstracción:

- Se destacan los detalles (normalmente pocos) de la especificación (el qué).
- Se ocultan los detalles (casi siempre numerosos) de la implementación (el cómo).

La abstracción

La abstracción, una de las herramientas que más nos ayuda a la hora de solucionar un problema, es un mecanismo fundamental para la comprensión de problemas y fenómenos que poseen una gran cantidad de detalles, su idea principal consiste en manejar un problema, fenómeno, objeto, tema o idea como un concepto general, sin considerar la gran cantidad de detalles que estos puedan tener.

El proceso de abstracción presenta dos aspectos complementarios:

1. Destacar los aspectos relevantes del objeto.
2. Ignorar los aspectos irrelevantes del mismo (la irrelevancia depende del nivel de abstracción, ya que si se pasa a niveles más concretos, es posible que ciertos aspectos pasen a ser relevantes).

De modo general podemos decir que la abstracción permite establecer un nivel jerárquico en el estudio de los fenómenos, el cual se establece por niveles sucesivos de detalles. Generalmente, se sigue un sentido descendente de detalles, desde los niveles más generales a los niveles más concretos.

Por ejemplo: los lenguajes de programación de alto nivel permiten al programador abstraerse del sin fin de detalles de los lenguajes ensambladores. Otro ejemplo, la memoria de la computadora es una estructura unidimensional formada por celdas y sin embargo trabajamos como si fuera única.

La abstracción nos brinda la posibilidad de ir definiendo una serie de refinamientos sucesivos a nuestro TDA y entiéndase bien que cuando decimos refinamientos sucesivos nos estamos refiriendo a la estrategia que se utiliza para descomponer un problema en subproblemas. Conforme evoluciona el diseño de software a cada nivel de módulos se representa un refinamiento en el nivel de abstracción. Esto es, incluir detalles que fueron obviados en un nivel superior, en un nivel más bajo de la jerarquía.

Veamos los diferentes tipos de abstracción que podemos encontrar en un programa:

1. **Abstracción funcional**: crear procedimientos y funciones e invocarlos mediante un nombre donde se destaca qué hace la función y se ignora cómo lo hace. El usuario solo necesita conocer la especificación de la abstracción (el qué) y puede ignorar el resto de los detalles (el cómo).

2. **Abstracción de datos**:

- **Tipo de datos**: proporcionado por los lenguajes de alto nivel. La representación usada es invisible al programador, al cual solo se le permite ver las operaciones predefinidas para cada tipo.
- **Tipos definidos**: por el programador que posibilitan la definición de valores de datos más cercanos al problema que se pretende resolver.
- **TDA**: para la definición y representación de tipos de datos (valores + operaciones), junto con sus propiedades.
- **Objetos**: Son TDA a los que se añade propiedades de reutilización y de compartición de código.

Si profundizamos más al mundo de la programación y sus conceptos, existen dos de estos conceptos que no se deben confundir, ellos son: tipo de datos y estructura de datos.

Un tipo de dato, en un lenguaje de programación, define un conjunto de valores que una determinada variable puede tomar, así como las operaciones básicas sobre dicho conjunto. Ahora veamos cómo se van relacionando estos conceptos. Los tipos de datos constituyen un primer nivel de abstracción, ya que no se tiene en cuenta cómo se implementan o se representan realmente la información sobre la memoria de la máquina. Para el usuario, el proceso de implementación o representación es invisible.

Veamos entonces qué son las estructuras de datos. Las estructuras de datos son colecciones de variables, no necesariamente del mismo tipo, relacionadas entre sí de alguna forma. Las estructuras de datos están caracterizadas por el tipo de dato de los elementos guardados en la estructura y por la relación definida sobre estos elementos.

Al nivel de las estructuras de datos son totalmente irrelevantes las operaciones sobre un elemento en particular, solamente tienen carácter relevante las operaciones que envuelvan la estructura de forma global.

La abstracción de datos es la característica de un sistema de bases de datos, que permite al usuario o programador operar con los datos sin necesidad de conocer detalles que para él no son de “importancia”, ofreciendo así una visión abstracta de estos. Para cumplir con tal fin se han definido diferentes niveles de abstracción:[1]

- Nivel Físico. Determina como están almacenados físicamente los datos (pistas, sectores, cilindros), representa el nivel más bajo.
- Nivel Lógico o Conceptual. Determina la organización de los archivos. Índices, llaves, orden de campos, relaciones, tipos de datos.
- Nivel de Vistas. Oculta parte de la información a los usuarios, es decir hace visible solo una parte de la base de datos.

Ejemplos de uso de TDA

Algunos ejemplos del uso de TDA en programación son:

- **Conjuntos**: implementación de conjuntos con sus operaciones básicas (unión, intersección y diferencia), operaciones de inserción, borrado, búsqueda...
- **Árboles Binarios de Búsqueda**: Implementación de árboles de elementos, utilizados para la representación interna de datos complejos. Aunque siempre se los toma como un TDA separado son parte de la familia de los grafos.

- **Pilas y Colas**: Implementación de los algoritmos FIFO y LIFO.
- **Grafos**: Implementación de grafos; una serie de vértices unidos mediante una serie de arcos o aristas.

Referencias

1. «Que es la abstracción de datos y modelos de datos» (https://web.archive.org/web/20110319032640/http://blogdecomputacion.com/blog/2010/08/21/que-es-la-abstraccion-de-datos-y-modelos-de-datos/). 21 de agosto de 2010. Archivado desde el original (http://blogdecomputacion.com/blog/2010/08/21/que-es-la-abstraccion-de-datos-y-modelos-de-datos/) el 19 de marzo de 2011. Consultado el 21 de enero de 2020.