Datos abstractos

Una herramienta útil para especificar las propiedades lógicas de un tipo de datos es el tipo abstracto de datos o ADT. Fundamentalmente, un tipo de datos es un conjunto de valores y un grupo de operaciones sobre tales valores. Este conjunto y estas operaciones forman una estructura matemática que se implementa usando una estructura particular de datos de hardware o software.

El término "tipo abstracto de datos" se refiere al concepto matemático básico que define el tipo de datos. Al definir un tipo de abstracto de datos como un concepto matemático, no interesa la eficiencia del espacio o del tiempo. Esos son aspectos de la implementación. De hecho, la definición
de un ADT no se relaciona en absoluto con los detalles de la implementación. Tal vez ni siquiera sea posible implementar un ADT particular en una parte de hardware especifica o usar un sistema de software determinado. Por ejemplo, ya hemos visto que no es posible implementar en forma universal un entero ADT. No obstante, debido a que especifica las propiedades matemática y lógica para los programadores que quieren usar los tipos de datos en forma correcta. de una estructura o tipo de datos, el ADT es una guía útil para quienes lo aplican y un recurso útil para los programadores que quieren usar los tipos de datos en forma correcta.

Un *Tipo de dato abstracto* (en adelante *TDA*) es un conjunto de datos u objetos al cual se le asocian *operaciones*. El TDA provee de una interfaz con la cual es posible realizar las operaciones permitidas, abstrayéndose de la manera en cómo estén implementadas dichas operaciones. Esto quiere decir que un mismo TDA puede ser implementado utilizando distintas estructuras de datos y proveer la misma funcionalidad.

Una vez que se ha elegido el algoritmo, la implementación puede hacerse usando las estructuras más simples, comunes en casi todos los lenguajes de programación: escalares, arreglos y matrices. Sin embargo, algunos problemas se pueden plantear en forma más simple o eficiente en términos de estructuras informáticas más complejas, como listas, pilas, colas, arboles, grafos, conjuntos. Por ejemplo, el TSP se plantea naturalmente en términos de un

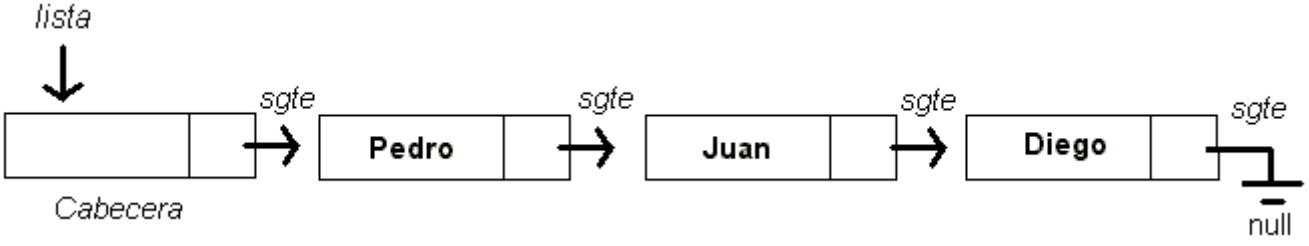
grafo donde los vértices son las ciudades y las aristas los caminos que van de una ciudad a otra. Estas estructuras están incorporadas en muchos lenguajes de programación o bien pueden obtenerse de librerías. El uso de estas estructuras tiene una serie de ventajas

- Se ahorra tiempo de programación ya que no es necesario codificar. Estas implementaciones suelen ser eficientes y robustas.
- Se separan dos capas de código bien diferentes, por una parte, el algoritmo que escribe el programador, y por otro las rutinas de acceso a las diferentes estructuras.
- Existen estimaciones bastante uniformes de los tiempos de ejecución de las diferentes operaciones.
- Las funciones asociadas a cada estructura son relativamente independientes del lenguaje o la implementación en particular. Así, una vez que se plantea un algoritmo en términos de operaciones sobre una tal estructura es fácil implementarlo en una variedad de lenguajes con una performance similar.

Tipos

Lista

Una *lista* se define como una serie de *N* elementos E_1, E_2, ..., E_N, ordenados de manera consecutiva, es decir, el elemento E_k (que se denomina *elemento k-ésimo*) es previo al elemento E_{k+1}. Si la lista contiene 0 elementos se denomina *lista vacía*.



-

Pila

Una *pila* (*stack* o *pushdown* en inglés) es una lista de elementos de la cual sólo se puede extraer el último elemento insertado. La posición en donde se encuentra dicho elemento se denomina *tope* de la pila. También se conoce a las pilas como *listas LIFO* (LAST IN - FIRST OUT: el último que entra es el primero que sale).

Listas enlazadas

En este caso no existe el problema de tener que fijar el tamaño máximo de la pila (¡aunque siempre se está acotado por la cantidad de memoria disponible!). La implementación es bastante simple: los elementos siempre se insertan al principio de la lista (apilar) y siempre se extrae el primer elemento de la lista (desapilar y tope), por lo que basta con tener una referencia al principio de la lista enlazada. Si dicha referencia es *null*, entonces la pila está vacía.

Dependiendo de la aplicación que se le dé a la pila es necesario definir qué acción realizar en caso de que ocurra *overflow* (rebalse de la pila) o *underflow* (intentar desapilar cuando la pila está vacía). Java posee un mecanismo denominado *excepciones*, que permite realizar acciones cuando se producen ciertos eventos específicos (como por ejemplo *overflow* o *underflow* en una pila).

En ambas implementaciones el costo de las operaciones que provee el TDA tienen costo *O(1)*.

Cola

Una *cola* (*queue* en inglés) es una lista de elementos en donde siempre se insertan nuevos elementos al final de la lista y se extraen elementos desde el inicio de la lista. También se conoce a las colas como *listas FIFO* (FIRST IN - FIRST OUT: el primero que entra es el primero que sale).

Al igual que con el TDA pila, una cola se puede implementar tanto con arreglos como con listas enlazadas.

Cola de Prioridad

Una *cola de prioridad* es un tipo de datos abstracto que almacena un conjunto de datos que poseen una llave perteneciente a algún conjunto ordenado, y permite *insertar* nuevos elementos y *extraer el máximo* (o el mínimo, en caso de que la estructura se organice con un criterio de orden inverso).

Es frecuente interpretar los valores de las llaves como prioridades, con lo cual la estructura permite insertar elementos de prioridad cualquiera, y extraer el de mejor prioridad.

Heaps

Un heap es un árbol binario de una forma especial, que permite su almacenamiento en un arreglo sin usar punteros.

Un heap tiene todos sus niveles llenos, excepto posiblemente el de más abajo, y en este último los nodos están lo más a la izquierda posible.

La característica que permite que un heap se pueda almacenar sin punteros es que, si se utiliza la numeración por niveles indicada, entonces la relación entre padres e hijos es:

Hijos del nodo j = {2*j, 2*j+1}
Padre del nodo k = floor(k/2)

Un heap puede utilizarse para implementar una cola de prioridad almacenando los datos de modo que las llaves estén siempre ordenadas de arriba a abajo (a diferencia de un árbol de búsqueda binaria, que ordena sus llaves de izquierda a derecha). En otras palabras, el padre debe tener siempre mayor prioridad que sus hijos.

Lista Enlazada

La Lista Enlazada Simple es la más fundamental estructura de datos basada en punteros, y del concepto fundamental de ésta derivan las otras estructuras de datos. Para solucionar un problema como el presentado anteriormente, necesitamos una estructura que, al contrario de los arreglos, sea capaz de modificar su capacidad, es decir, que maneje los datos de forma dinámica. Para lograr esto, nace la idea de lista enlazada. Un arreglo asigna memoria para todos sus elementos ordenados como un sólo bloque. En cambio, la lista enlazada asigna espacio para cada elemento por separado, en su propio bloque de memoria, llamado nodo. La lista conecta estos nodos usando punteros, formando una estructura parecida a la de una cadena.

Árbol

Los árboles son una de las estructuras de datos no lineales más utilizada. Sirve para representar estructuras de información jerárquicas y direcciones o etiquetas de una manera organizada.

Lista Circular

Es una lista lineal en la que el último nodo a punta al primero.
Las listas circulares evitan excepciones en las operaciones que se realicen sobre ellas. Cada nodo siempre tiene uno anterior y uno siguiente.

Lista Doble Enlace

En este tipo de listas cada nodo ha de tener 3 campos: un campo que tiene el dato, otro que tiene la referencia del siguiente nodo y otro que tiene la referencia del nodo anterior. Los nodos de la lista son enlazados por medio de los campos enlaces. El último nodo de una lista de doble enlace no tiene siguiente y el primer nodo de la lista no tiene anterior.

Doble Cola

Esta estructura es una cola bidimensional en que las inserciones y eliminaciones se pueden realizar en cualquiera de los dos extremos de la bicola.

Existen dos variantes de la doble cola:

- Doble cola de entrada restringida.
- Doble cola de salida restringida.

La primera variante sólo acepta inserciones al final de la cola, y la segunda acepta eliminaciones sólo al frente de la cola.

Una bicola o cola bidirecional, es una lista lineal en la que los elementos que se pueden añadir o quitar por cualquier extremo. Existen dos:

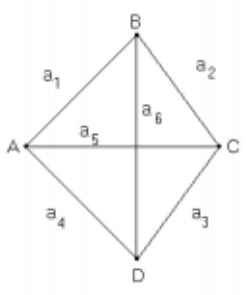
- Bicolas de entrada restringida: Son aquellas donde la inserción sólo se hace por el final, aunque podemos eliminar al principio o al final.
- Bicolas de salida restringida: Son aquellas donde sólo se elimina por el final, aunque se puede insertar al principio y al final.

Grafos

Consiste en un conjunto N de elementos llamados nodos (n), también conocidos como vértices o puntos; Así como también por un conjunto A de líneas que unen un elemento con otro y se denominan aristas (a), cada arista se identifica por un único par ordenado [u, v], donde u y v son los extremos adyacentes a “a”.

Grafo conexo.- Es conexo, si y solo sí existe un camino simple entre cualquier par de nodos de G.

Grafo completo.- cuando cada nodo de G es adyacente a todos los demás nodos.

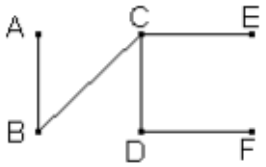


*Un grafo completo de **n** nodos tiene:*

n(n-1)/2 aristas

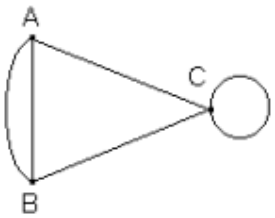
Ejemplo: *n = 4 nodos* ----->
Aplicando fórmula = 6 aristas

Grafo árbol.- En el cual existe un único camino simple entre cada dos nodos.



Grafo etiquetado.- Si sus aristas tienen datos asignados a los cuales se les denomina pesos.

Multígrafo.- Es aquel que permite aristas múltiples es decir que dos nodos tengan más de una arista y permite los bucles es decir cuando mas de una arista tiene los mismos



GRAFO DIRIGIDO.- Es aquel en el cual cada arista tiene una dirección asignada, es decir cada arista se identifica con un par ordenado (u, v) en vez del par desordenado [u, v]

Esto significa que:

- Cada nodo empieza en "u" y termina en "v"
- "u" es el origen y "v" es el destino
- "u" es el predecesor y "v" es el sucesor de "u"
- "u" es adyacente hacia "v" y "v" es adyacente desde "u"

grado de salida.- Es el número de aristas que empiezan en el nodo "u"

grado de entrada.- Es el número de aristas que terminan en el nodo "u"

nodo fuente.- Si tiene grado de salida positivo y grado de entrada nulo.

nodo sumidero.- Si tiene grado de salida nulo y grado de entrada positivo.

Referencias

1. Estructura de datos: Referencia práctica con orientación a objetos, Román Martínez, THOMSON LEARNING, 2004, México.
2. Estructuras de datos con C y C++, Yedidyah Langsam, PEARSON Educación, 1997, México, Pag 13.
3. Estructura de datos y algoritmos, Perla Sañas y Sergio Matig, 2005, Pag 1.
4. Estructura de datos y algoritmos, Perla Sañas y Sergio Matig, 2005, Pag 2.
5. Estructuras de Datos Abstractas en Lenguaje Java, Chile, Carlo Casorzo G. Pag 7.
6. Estructuras de Datos Abstractas en Lenguaje Java, Chile, Carlo Casorzo G. Pag 16.
7. Estructuras de Datos Abstractas en Lenguaje Java, Chile, Carlo Casorzo G. Pag 21.
8. Estructuras de Datos Abstractas en Lenguaje Java, Chile, Carlo Casorzo G. Pag 25.
9. Estructuras de datos en Java, Luis Felipe Wanumen Silva, UDEditorial, 2017, Bogotá, Pag 93.
10. Estructuras de datos en Java, Luis Felipe Wanumen Silva, UDEditorial, 2017, Bogotá, Pag 103.
11. Estructuras de Datos en Java, Cristian Denis Mamani Torres, AlgoritmoD, Pag 109.
12. Estructuras de Datos en Java, Cristian Denis Mamani Torres, AlgoritmoD, Pag 110.
13. Manual de Algoritmos y Estructura de Datos, Ing. Hugo Caulli Gimondi, Chimbote, 2009, Perú, Pag 35.
14. Manual de Algoritmos y Estructura de Datos, Ing. Hugo Caulli Gimondi, Chimbote, 2009, Perú, Pag 52.