**ESTRUCTURA DE DATOS**

1. **Estructuras Estáticas**
   1. **Arreglos (Arrays)**

Colección ordenada de datos, que puede guardar diferentes tipos de datos. Son eficientes para “Obtener o Leer” los datos; sin embargo, para “Agregar o Borrar” (aunque se puede) no lo son.

|  |  |  |
| --- | --- | --- |
| **Acciones** | **Forma** | **Explicación** |
| **Tipos de Datos** | A(n) | |  |  |  | | --- | --- | --- | | “Hola” | 33 | True | | Cadena | Números | Booleanos | |
| 🗹 **Obtener** | A(1) | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | A(i) | H | O | L | A |  | | i | 0 | 1 | 2 | 3 | 4 | |
| 🗷 **Agregar** | A(n) |
| 🗷 **Eliminar** | A(n) |
| **Ejemplo de Agregar elemento** | **En Memoria** | **Grados de temperatura:**  Cuando se agrega un dato se crea una copia en memoria para agregarlo. Por esta razón, es ineficiente.   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 23 | 25 | 20 | 28 | 15 | X |  |  | | 23 | 25 | 20 | 12 | 28 | 15 |  |  | |

1. **Estructuras Dinámicas**
   1. **Punteros**

Sin entrar en detalles, un puntero o apuntador es una variable estática de tipo entero usada para contener la dirección de memoria de otra variable (variable que apunta a otra variable). Se utilizan principalmente para realizar operaciones con estructuras dinámicas de datos, es decir, estructuras creadas en tiempo de ejecución, siendo su objetivo el de permitir el manejo o procesamiento (creación, acceso, eliminación, etc.) de estas estructuras de datos.

Imagen que contiene Gráfico

Descripción generada automáticamente

**Operaciones con punteros**

* **Definición:** Nombre Puntero es un puntero a Tipo\_Elemento.
* **Inicialización:** Nombre Puntero = Nulo.
* **Creación:** Reservar(Nombre Puntero).
* **Destrucción:** Liberar(Nombre Puntero).
* **Acceso:** Nombre Puntero (accedemos a la dirección de memoria), Val-Nombre Puntero (accedemos al valor contenido en la dirección de memoria).
  1. **Estructuras Dinámicas Lineales**
     1. **Listas o Listas Enlazadas (Linked List)**

Una lista es una estructura dinámica de datos que contiene una colección de elementos homogéneos (del mismo tipo) de manera que se establece entre ellos un orden. Es decir, cada elemento, menos el primero, tiene un predecesor, y cada elemento, menos el último, tiene un sucesor.

Podemos distinguir, atendiendo a la organización de los nodos, entre:

* **Listas simplemente enlazadas:** Cada nodo tiene un campo que apunta al siguiente nodo.
* **Listas doblemente enlazadas:** Cada nodo dispone de un puntero que apunta al siguiente nodo, y otro que apunta al nodo anterior.

Otra distinción puede ser:

* **Listas lineales:** Son listas que tienen un comienzo y un final.
* **Listas circulares:** En estas listas el último elemento apunta al primero, por lo tanto, podríamos estar recorriéndolas siempre, ya que no tienen final.

|  |  |  |
| --- | --- | --- |
| **Acciones** | **Forma** | **Explicación** |
| 🗷 **Obtener** | **Nodo (dato) y puntero** | 32  60  15  9 |
| 🗹 **Agregar** |
| 🗹 **Eliminar** |
| **Ejemplo de Agregar y Eliminar Elemento** | **En Memoria** | **Grados de temperatura:**  Cuando se busca obtener o leer un dato, tiene que empezar por el inicio o final para encontrarlo. Por esta razón, es ineficiente.   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 32 |  |  |  | 60 |  |  |  | 15 | |  |  |  | 3 |  |  | 9 |  |  | |
| **Tipos** | Lista simplemente enlazada  Lista doblemente enlazada |

* + - 1. **Pilas (Stacks)**

Una pila es una estructura de datos dinámica formada por un grupo ordenado de elementos homogéneos (todos del mismo tipo) en la que los elementos sólo pueden ser añadidos o eliminados por un extremo llamado cabecera o cima (top) de la pila. De esta forma, los elementos se extraen en orden inverso al seguido en el proceso de inserción. Por ello, una pila es considerada una estructura de datos LIFO (Last In First Out), esto es, que el último elemento que entra es el primero que sale.

Recibe su nombre por su similitud con una pila de objetos; si tenemos una pila de 5 libros, siempre cogemos el de arriba, que es el último que se dejó. Si queremos leer un libro que está en cuarto lugar debemos coger primero el quinto libro y a continuación el cuarto.

Las operaciones básicas que podemos realizar sobre una pila son dos: almacenamiento (push) y recuperación (pop). Para manejar una estructura de datos tipo pila, un programador debe definir un conjunto de operaciones que permitan al usuario acceder y manipular los elementos en ella almacenados. Generalmente, cada una de estas operaciones suele recibir un nombre fijo e identificativo de la operación que realiza, siendo la terminología utilizada más común la mostrada a continuación:

* **Meter o Apilar:** Se denomina así a la operación que añade un elemento a una pila.
* **Sacar o Desapilar:** Operación que saca un elemento de la pila.
* **InicializarPila:** Una vez creada la pila y antes de usarla, esta debe quedar inicialmente vacía.
* **PilaVacia:** Operación que devuelve un valor de tipo lógico para comprobar si la pila se encuentra vacía antes de sacar un elemento.
* **PilaLlena:** La usaremos en aquellos casos en que sea necesario saber si la pila se encuentra llena antes de añadir un nuevo elemento.
* **Cima:** Útil para obtener el último elemento que fue añadido a la pila, este es, el primero en salir de ella.

Sus acciones más importantes son: push() [agregar], pop() [eliminar] y peek() [encontrar último elemento de la pila]; un elemento.

|  |  |  |
| --- | --- | --- |
| **Acciones** | **Forma** | **Explicación** |
| 🗹 **peek()** | **- Apilar (push())**  **- Desapilar(pop())**  **- Encontrar elemento de la pila (peek())** | Gráfico  Descripción generada automáticamente |
| 🗹 **push()** |
| 🗹 **pop()** |

* + - 1. **Colas (Queves) o Filas**

Una cola es un grupo ordenado de elementos del mismo tipo, en la cual dichos elementos se añaden por un extremo (Final) y se quitan por el otro extremo (Frente). Esto significa que los elementos se sacan en el mismo orden en el que fueron insertados o introducidos en la cola, siendo por ello considerada como una estructura de datos FIFO (First In First Out), es decir, que el primer elemento en entrar es el primer elemento en salir.

A modo de ejemplo, una estructura FIFO funciona exactamente igual que una cola en un establecimiento, cuando llegas lo primero que debes hacer es pedir la vez, en ese momento te sitúas detrás del último que llegó, esto es, al final de la cola. A la hora de atender, el dependiente atiende a quién ocupa la primera posición de la cola, y así sucesivamente hasta que te toque a ti. Otros ejemplos del uso de colas pueden ser una cola para la impresión de documentos, o en un videojuego la animación de varios elementos en movimiento dentro de la pantalla, así se crea una cola para ir animando uno a uno los personajes del juego, de forma que cuando avanzamos alguno una posición se manda otra vez al final de la cola, lo que da la sensación de que se están moviendo todos simultáneamente.

Para manejar una estructura de datos de tipo cola, un programador debe definir un conjunto de operaciones que permitan al usuario acceder y manipular los elementos almacenados en ella. La terminología más común utilizada para referenciar este conjunto de operaciones es la mostrada a continuación:

* **InicializarCola:** Nos permite dejar inicialmente vacía la cola una vez creada.
* **Encolar:** Permite añadir un elemento al final de la cola.
* **Desencolar:** Se usará para sacar un elemento de la cola.
* **ColaVacia:** Devolverá cierto si la cola está vacía antes de sacar un elemento de la misma.
* **ColaLlena:** Sólo en aquellos casos en los que sea necesario determinar si la cola se encuentra llena antes de añadir un nuevo elemento debido a la implementación utilizada (sólo en el caso de implementación con tablas), emplearemos esta operación de tipo lógico.

Sus acciones más importantes son: enqueve() [agregar], requeve() [eliminar] y front() [encontrar primer elemento de la cola]; un elemento.

|  |  |  |
| --- | --- | --- |
| **Acciones** | **Forma** | **Explicación** |
| 🗹 **front()** | **- Apilar (push())**  **- Desapilar(pop())**  **- Encontrar elemento de la pila (peek())** | Imagen que contiene Tabla  Descripción generada automáticamente  Implementación de una cola usando punteros (creación)  Interfaz de usuario gráfica, Aplicación, Sitio web  Descripción generada automáticamente |
| 🗹 **enqueve()** |
| 🗹 **requeve()** |

* + 1. **Mapas o Diccionarios (Hash Maps o Hash Tables)**

Son estructuras de datos que almacenan los datos en memoria, bajo el concepto de paridad “Clave” y “Valor”.

|  |  |  |
| --- | --- | --- |
| **Acciones** | **Forma** | **Explicación** |
| 🗹 **Obtener** 🗹 | **Par (Pair):**  **- Clave (Key)**  **- Valor (Value)** | |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | Value |  | “Argentina” |  | “Uruguay” |  | “Venezuela” | | Key |  | “Buenos Aires” |  | “Montevideo” |  | “Caracas” | |
| 🗹 **Agregar** |
| 🗹 **Eliminar** |
| **Ejemplo de organización de los datos** | **En Memoria** | **Países y capitales:**  Cuando se obtiene, agrega o elimina un elemento; se hace con la llave, no se mueven los elementos y esto hace que, sean muy eficientes en las acciones anteriores.   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  |  | “Venezuela” | | “Uruguay” |  |  |  | “Argentina” |  |  |  |   “Caracas”  “Montevideo”  “Buenos Aires” |

* 1. **Estructuras dinámicas no lineales (Árboles y Grafos)**

Hasta el momento, todo lo que hemos visto han sido estructuras de datos lineales, pero en ocasiones son necesarias otro tipo de estructuras de datos jerarquizadas como pueden ser los árboles o los grafos.

* + 1. **Árboles (Tree)**

Un árbol es una estructura de datos jerarquizada y no lineal constituida por un conjunto de elementos homogéneos, es decir del mismo tipo, que se caracterizan porque establecen una jerarquía entre los elementos que la forman. Esta relación jerárquica entre sus nodos se establece de manera que todos los nodos que componen el árbol tienen un único nodo padre, que es el nodo que le precede en la jerarquía. La única excepción es el nodo raíz, que se puede definir como aquel nodo que no tiene padre. Todos los árboles tienen una única raíz.

Cada nodo puede tener 0, 1, o más de un nodo hijo. Un nodo hijo es aquel que le sucede en la jerarquía. Si un nodo no tiene ningún hijo se le llama nodo terminal u hoja. El número de hijos que tiene un nodo es lo que denominaremos grado.

Un árbol se puede caracterizar por:

* **Profundidad:** Máximo grado que alcanza el mismo.
* **Peso:** Número de hojas del árbol.

Los árboles se clasifican en función del número máximo de sucesores o descendientes de un nodo, de forma que un árbol donde cada nodo tiene un máximo de dos sucesores se denomina árbol binario (son los más usados); si tiene un máximo de tres sucesores se denomina árbol ternario y así sucesivamente.

|  |  |  |
| --- | --- | --- |
| **Acciones** | **Forma** | **Explicación** |
|  | **- Nodo Principal (root)**  **- Nodo (dato)**  **- Puente (conecta Nodo Padre e Hijo)** | Ejemplos gráficos de árboles binarios |
|  |
|  |
| **Organizar números en un Árbol Binario** | **Los números menores al primero van al lado izquierdo del árbol y; los números mayores van al lado derecho del árbol.** |  |

* + 1. **Grafos (Graphs)**

Un grafo es una estructura de datos dinámica muy similar a los árboles formado por un conjunto de nodos y otro conjunto de arcos. Cada arco agrupa a dos nodos que pueden ser el mismo. Cada arco puede tener peso, coste o distancia. Además de arcos pueden estar orientados o no estarlo. En caso de estar orientados se representan mediante una flecha, en caso contrario se representan por un segmento.

Un grafo se puede recorrer igual que un árbol, bien por profundidad o bien por anchura. Algunos algoritmos importantes, como el de Djikstra, son desarrollados mediante grafos. Resolver por ejemplo un problema de distancias entre ciudades es típico de grafos. Aquí cada vértice sería una ciudad y cada arco una carretera entre dos ciudades.

|  |  |  |
| --- | --- | --- |
| **Acciones** | **Forma** | **Explicación** |
|  |  | Grafo de distancias entre ciudades |
|  |
|  |

1. **Mapas o Diccionarios (Hash Maps o Hash Tables)**