Estructura de datos

CESAR AUGUSTO LUNA LOPEZ

Red Tercer Milenio

ESTRUCTURA DE DATOS

ESTRUCTURA DE DATOS

CESAR AUGUSTO LUNA LOPEZ

RED TERCER MILENIO

AVISO LEGAL

Datos para catalogación bibliográfica

César Augusto Luna López

Estructura de datos

ISBN 978-607-733-129-2

Primera edición: 2012

Eduardo Durán Valdivieso

DIRECTORIO

ÍNDICE

INTRODUCCIÓN

En la actualidad, la eficiencia de un programa informático va de la mano con las técnicas de programación que se emplean en su desarrollo, partiendo desde la elaboración de diagramas de flujo de datos, hasta la escritura de los códigos para el desarrollo del software. Lo anterior busca el acceso a los datos de la información de una manera ordenada mediante instrucciones válidas, empleando una secuencia lógica.

La estructura de datos se refiere a un conjunto de técnicas que aumentan considerablemente la productividad del programa, reduciendo en elevado grado, el tiempo requerido para escribir, verificar, depurar y mantener los programas. El término estructura de datos hace referencia a un conjunto de datos que, por medio de un nombre, identifican un espacio en memoria, teniendo ciertas características como la organización y estructuración, permitiendo realizar operaciones definidas en ellas. Las estructuras de datos pueden ser de dos tipos:

- Estructuras de datos estáticas (las que tienen un tamaño definido).
- Estructuras de datos dinámicas (en las cuales su tamaño puede ser cambiado en tiempo de ejecución).

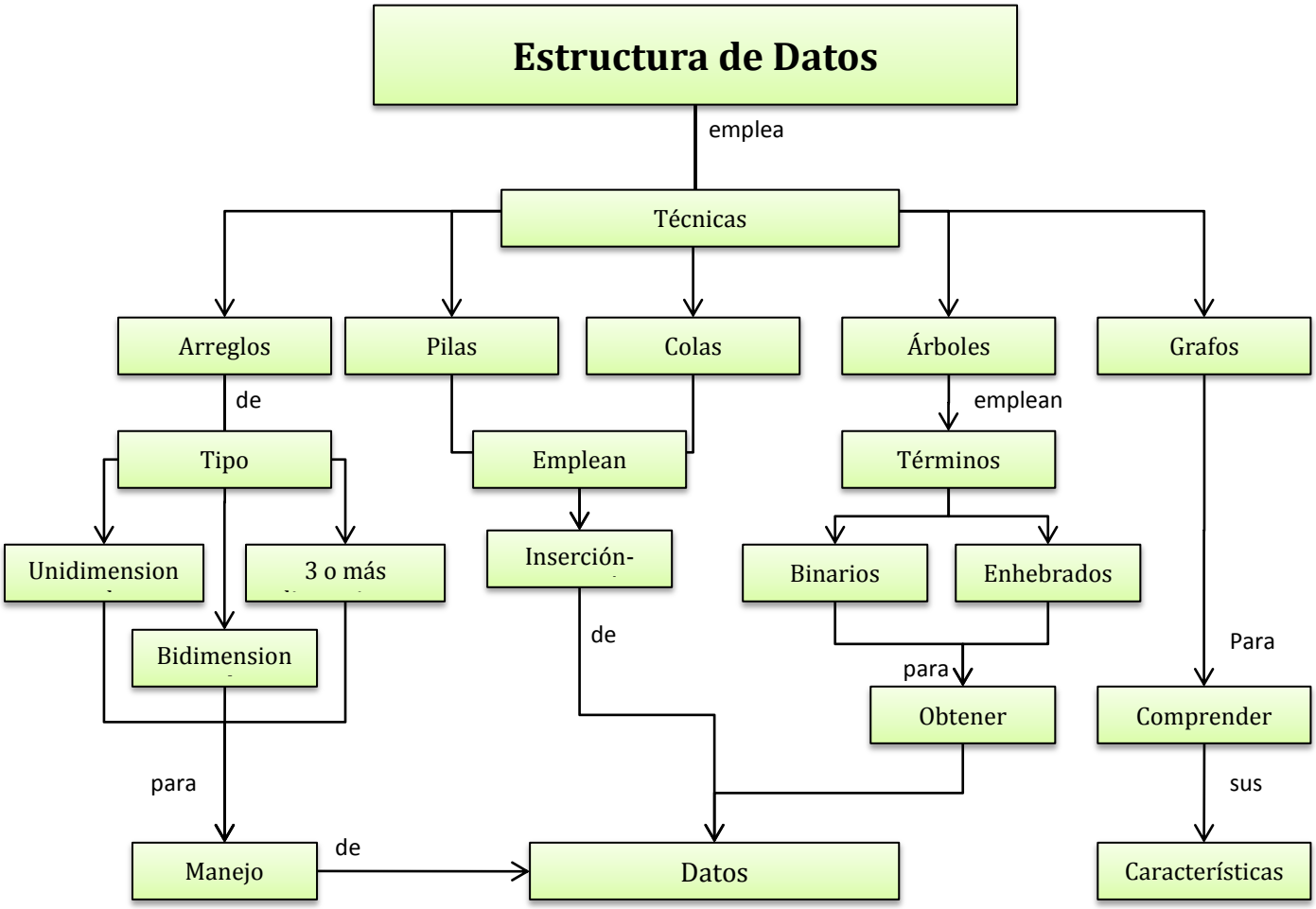
La presente obra comienza su estudio, con las estructuras de datos estáticas, analizando el concepto y fundamento de los arreglos, sus métodos para el manejo de datos, sus variantes que pueden dar origen a estructuras de arreglos de una o más dimensiones.

En la unidad dos analizaremos el concepto de las pilas y colas, los métodos en que se manipulan los datos para insertarlos o eliminarlos, así como sus reglas de uso.

En la unidad tres combinaremos lo aprendidos en las anteriores unidades para lograr realizar prácticas más complejas, donde involucren conceptos y métodos que permitan ordenar datos, empleando arreglos, o bien para realizar búsquedas de información a través de algoritmos computacionales. Las últimas unidades abordarán los temas de árboles y

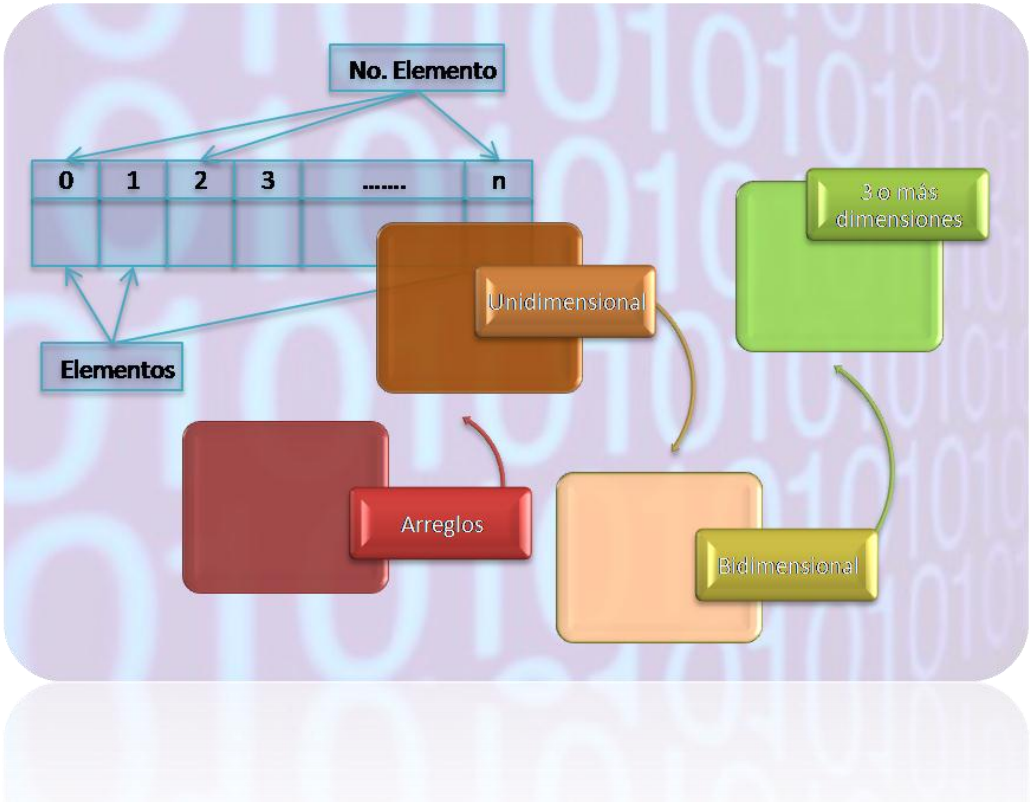
grafos, que son estructuras dinámicas que nos permitirán trabajar con almacenamientos secundarios. Asimismo, se introducirá sobre el uso de los grafos, su empleo y creación dentro de un lenguaje de programación.

MAPA CONCEPTUAL



UNIDAD 1

ARREGLOS

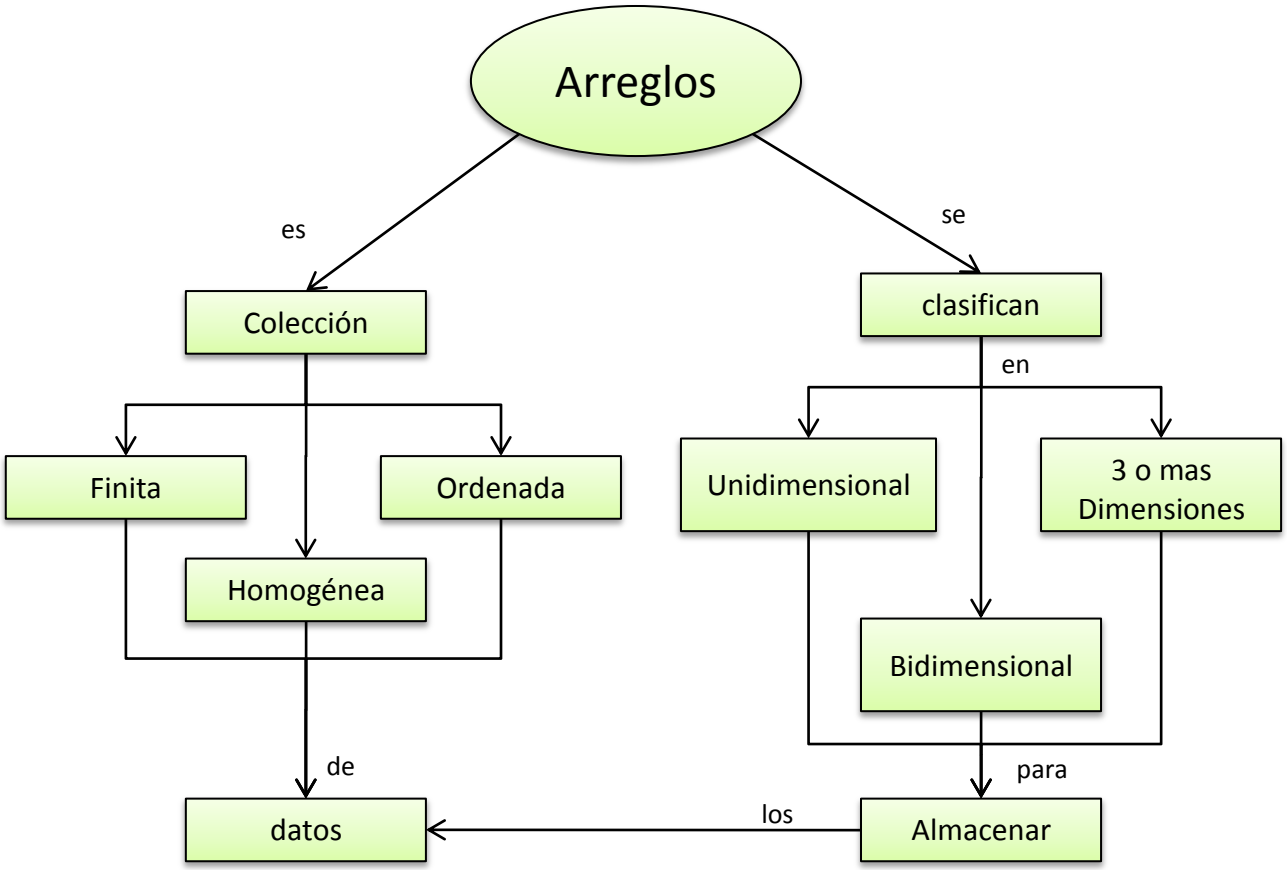


OBJETIVO

Analizar el uso e implementación de los arreglos para la resolución de problemas de datos estructurados, mejorando el tiempo de desarrollo y eficacia de los sistemas.

TEMARIO

MAPA CONCEPTUAL



INTRODUCCIÓN

En esta unidad conoceremos uno de los tipos de datos estructurados más empleados dentro de la programación, los arreglos. La utilidad de estos es trascendental al momento de codificar los programas, debido a que dentro de un programa tendemos a emplear una gran cantidad de variables, esto puede ser un poco ortodoxo debido a que si necesitamos 10, 20 o 100 variables para un mismo tipo de datos, la manipulación de estos se vuelve más complicada.

¿Entonces como resolvemos el manejar tantas variables? Aquí es donde se emplean los arreglos, para formar una estructura más definida, más fácil de procesar e implementar en nuestros códigos.

Partiremos de la conceptualización de los arreglos, como dependiendo de la necesidad pueden emplearse de una, dos o más dimensiones. Como escribir y leer sus datos, así como ejemplos de la codificación en el lenguaje Java.

1.1. CONCEPTOS

Un arreglo es una estructura de datos lineal, “un arreglo puede definirse como un grupo o una colección finita, homogénea y ordenada de elementos.”[1] Donde finita significa que debe tener un límite de elementos, homogénea que los elementos deben de ser del mismo tipo de datos, y ordenada porque se puede conocer cuál es el primer elemento, los subsiguientes o el último.

En la siguiente gráfica puede observarse cómo se representa un arreglo y se identifican las partes que lo integran.

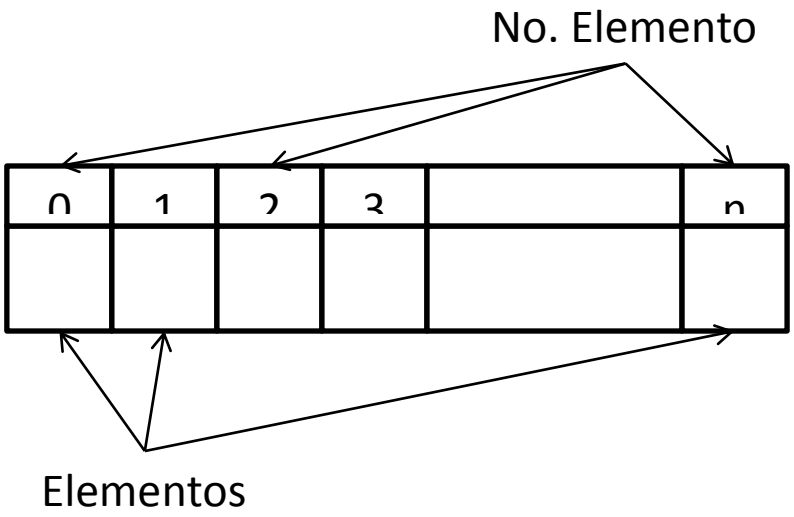


Fig.1.1. Representación de un arreglo

Dentro del uso de los arreglos, se puede hacer referencia a un número de elemento para introducir, actualizar o extraer valores, ese Número de elemento se conoce como índice y especifica la posición del elemento en el arreglo.

Para referirse a una posición del arreglo, es necesario conocer el *nombre* del arreglo y su *índice*, por ejemplo, tomando la figura 1.2., tenemos un arreglo llamado “Datos” y necesitamos extraer el valor “77”, es necesario especificar: Datos [3]

[1] Osvaldo Cairó /Silvia Guardati , Estructuras de datos, pág. 4

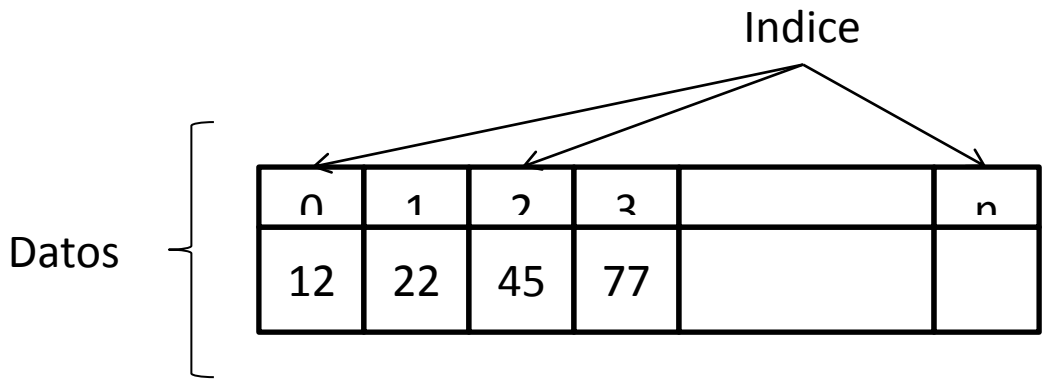


Fig. 1.2. Arreglo Datos

Según su estructura los arreglos se clasifican en:

- Arreglos unidimensionales o de una dimensión.
- Arreglos bidimensionales o de dos dimensiones.
- Arreglos tridimensionales o de tres o más dimensiones.

1.2. ARREGLOS UNIDIMENSIONALES

El concepto de arreglo empleado en el tema anterior, se aplica también para referirse a los arreglos de dimensión, es decir, es un tipo de datos estructurado compuesto de un número de elementos finitos, homogéneos y ordenados.

Los elementos del arreglo se almacenan en posiciones contiguas de memoria, a cada una de las cuales se puede acceder directamente mediante su índice. La forma de acceder a un arreglo de una dimensión es directa, pues se puede especificar el índice del elemento sin necesidad de conocer el contenido de los elementos contiguos.

Esto se puede ilustrar mejor con el siguiente ejemplo: supóngase que desea capturar las edades de un grupo de 100 personas, para conocer cuántas están sobre el promedio de edad.

Este problema se podría resolver empleando 100 variables simples para almacenar las edades; luego, se calcula el promedio, y por último se comparan las 100 variables para determinar cuántas personas están sobre el promedio. Este método consumirá muchos recursos del sistema (principalmente por la creación de tantas variables), y representa la complejidad de utilizar en la programación sólo datos simples.

Definición de un arreglo. La forma correcta de resolver el anterior problema, sería empleando un arreglo. ¿Pero cómo se definen?, esto depende principalmente del lenguaje de programación que utilicemos, para cuestiones de estudio, la mayoría de los autores sobre este tema emplean una sintaxis generalizada de la siguiente forma:

Nombre_Arreglo = ARREGLO [Limite_inferior . . Limite_Superior] de TIPO

Donde:

Arreglo. Es la palabra reservada para crear los arreglos

Nombre_Arreglo. Representa el Nombre que emplearemos para referirnos al arreglo

Limite_Inferior y Limite_Superior. Representan el número Índice para indicar el inicio y fin del arreglo, es decir, un rango, por ejemplo del [1 .. 3] o [5 .. 20].

Tipo. Es el tipo de datos que almacenará ese arreglo, hay que recordar que los arreglos son homogéneos y pueden ser cualquier tipo ordinal (cadena, entero, booleano, real, etc.)

Como ya se ha mencionado, manejaremos la sintaxis empleado en el lenguaje JAVA, donde los arreglos se definen de la siguiente forma:

Tipo Nombre_Arreglo [ ] = new Tipo [Tamaño_Arreglo];

Note como en este lenguaje sólo se emplea Tamaño_Arreglo, para definir el tamaño y no el rango, esto es debido a que JAVA emplea como primer índice el 0. Así que por deducción indicando el Tamaño_arreglo, el lenguaje emplea el rango 0 al Tamaño_Arreglo-1.

Retomando el ejemplo de las edades, para almacenar los valores basta con definir un arreglo de la siguiente forma:

int Edad [ ] = new Edad [100];

La representación gráfica del anterior arreglo, se presenta en la siguiente ilustración:

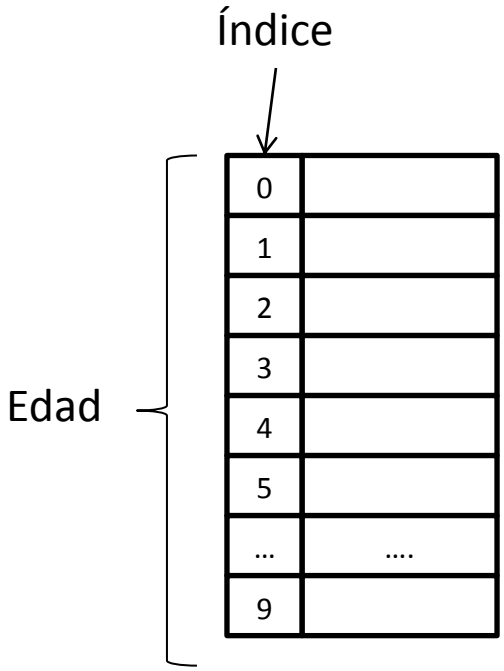


Fig. 1.3. Arreglo de 100 posiciones

En el proceso de escritura en un arreglo, se debe escribir un valor para cada uno de sus componentes de la siguiente manera:

Nombre_ Arreglo [Indice] = Valor

Empleando el lenguaje JAVA quedaría:

Edad [0] = 33;

Almacenando en el arreglo Edad en la posición 0 el valor de 33. Para asignar los valores es conveniente que se integre un ciclo para que la lectura sea más automatizada.

```
Repetir con I desde 1 hasta N
Leer E
Escribir Arreglo [ I ] = E
```

Para actualizar el contenido de un componente del arreglo, sólo basta con reasignarle el valor, como una variable simple, el anterior valor se elimina para dar paso al nuevo valor.

Otra forma de crear arreglos, es empleando la asignación directa de la forma:

Tipo Nombre_Arreglo [ ] = {Valor1, Valor2, Valor3…Valor_n}

Ejemplos:

String Cadenas [ ] ={“Juan”, ”Pedro”, “Tomas”, “Raul”};

Esto creará automáticamente un arreglo con la cantidad de registros especificados, es decir, creará un arreglo llamado Cadenas con 4 componentes, sustituyendo do instrucciones String Cadenas[ ] = new String [4] y la asignación de los datos.

Dentro de las estructuras de los arreglos unidimensionales, podemos llevar a cabo las siguientes operaciones:

- Lectura.
- Escritura.
- Actualización.
- Ordenación.
- Búsqueda.

Lectura. Esta operación se refiere al hecho de leer los datos de un registro contenido en un arreglo, permitiendo asignar el valor a una variable.

X ← Arreglo [i] donde i representa la posición del arreglo

Escritura. Esta operación se refiere al hecho de asignarle valor al registro contenido en un arreglo.

Arreglo [i] ← valor

Actualización. En este proceso, las operaciones consideradas como actualización, son los procesos de eliminar, insertar y modificar datos, tomando en cuenta si los datos están ordenados o no.

Ordenación. Este proceso consiste en reordenar los datos del arreglo tomando un criterio, por ejemplo, ordenar los datos numéricos de mayor a

menor, ordenar las cadenas alfabéticamente, etc. Los métodos de ordenamiento serán abordados con detalle más adelante.

Continuando con el ejemplo de las edades, se presenta el código para la solución en pseudocódigo y Java:

```
Inicio
        Definir  Max← 100
        definir arreglo “Arreglo_Edad”
        desde i=0 hasta i < Max
                leer Edad;
                Arreglo_edad[ i ] ←edad
                Suma ←Suma + Arreglo_edad [ i ]
        Fin_desde
        Promedio←Suma/Max
        Des i=0 hasta i < Max
                Tempo← Arreglo_Edad[i]
                Si Tempo > Promedio entonces
                        Arriba_Prom=Arriba_Prom + 1
                Fin_Si
        Fin_desde
        Imprimir Arriba_Prom
Fin_Inicio
```

```
1    import javax.swing.JOptionPane;
2
3   public class ArregloUni {
4       public static void main(String[] args) {
5           int Max=100, Suma=0, Promedio=0,Tempo=0, Arriba_Prom=0;
6          String Edad;
7           int Arreglo_Edad[]=new int[Max];              // Declaraccion del arreglo
8                for (int i=0; i<Max;i++){
9                Edad= JOptionPane.showInputDialog("Edad:");
10              Arreglo_Edad [ i ] = Integer.parseInt (Edad);
11                   Suma += Arreglo_Edad [ i ];
12               }
```

```
13              Promedio=Suma/Max;
14
15          for (int i=0; i<Max;i++){
16                  Tempo=Arreglo_Edad[i];
17                  if (Tempo>Promedio){
18                      Arriba_Prom++;
19                  }
20          }
21
22      JOptionPane.showMessageDialog(null, "El número de edades arriba
        del promedio es "+Arriba_Prom);
23
24    }
25  }
```

Donde se puede observar la declaración del arreglo en la línea 7

int Arreglo_Edad [ ] =new int [Max];

Además, la asignación de los valores en la línea10, donde se convierte la cadena a entero para asignarla al arreglo.

Arreglo_Edad [i] = Integer.parseInt (Edad);

Y por último, la lectura de los valores del arreglo en la línea 16

Tempo=Arreglo_Edad[i];

ACTIVIDAD DE APRENDIZAJE

1.- Desarrolla un programa que permita la captura de las ventas de un Comisionista de forma mensual, mostrando al final la venta Promedio, la venta más alta y la más baja.

1.3. ARREGLOS BIDIMENSIONALES

“Un arreglo bidimensional es un conjunto de datos homogéneo, finito y ordenado, donde se hace referencia a cada elemento por medio de dos

índices. El primero se utiliza generalmente para indicar renglón, y el segundo para indicar columnas. También puede definirse como un arreglo de arreglos."[2]

Un arreglo bidimensional (también conocido como tabla o matriz), es un arreglo con dos índices, al igual que los vectores que deben ser ordinales.

Es común destacar que el empleo de los arreglos de dos dimensiones se emplea para la construcción de tablas, en la cual están representadas por filas y columnas. Como se mencionó, para localizar o almacenar un valor en el arreglo se deben especificar dos posiciones (dos subíndices), uno para la fila y otro para la columna.

Arreglo [Fila, Columna]

La definición de un arreglo bidimensional quedaría con la siguiente sintaxis:

Tipo Nombre_Arreglo [ ] [ ] = new Tipo [Tam_Arreglo_Fila]
[Tam_Arreglo_Columna];

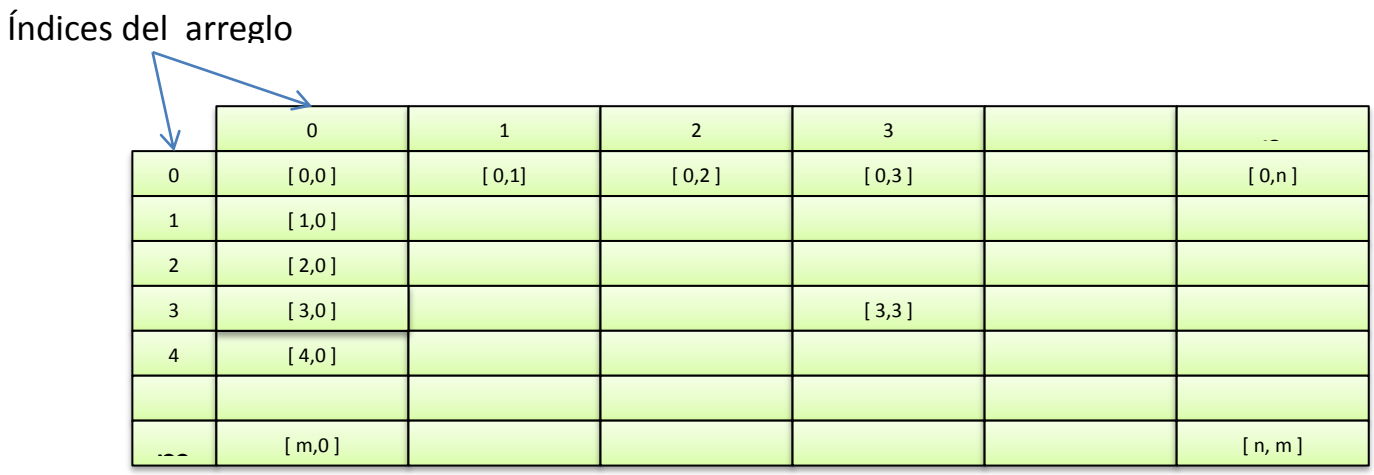


Fig. 1.4. Representación gráfica de un arreglo bidimensional

Donde Tam_Arreglo_Fila se define el número de renglones que tendrá el arreglo y con Tam_Arreglo_Columna se declara la cantidad de columnas que contendrá el arreglo

Las operaciones empleadas en un arreglo como la lectura, escritura, actualización, ordenamiento y búsqueda, seguirán el mismo método de los arreglos unidimensionales, con la diferencia de los dos índices.

[2] Osvaldo Cairó /Silvia Guardati, *Estructuras de datos*, p. 19.

Por ejemplo, necesitamos una matriz donde se almacenen las calificaciones de 30 personas, conteniendo cinco materias y su matrícula. La estructura del arreglo quedaría de la siguiente forma:

```
int Registros [ ] [ ] = new int [ 30 ][ 6 ];
```

Otra variante sería la siguiente:

```
String Datos[]=
{“Matricula”,”;Materia1”,”Materia2”,”Materia3”,”Materia4”,”Materia5”};
int Registros [ ] [ ] = new int [ 30 ][ Datos.length ];
```

El ejemplo completo sería el siguiente:

```
1   import javax.swing.JOptionPane;
2
3   public class Bidi {
4
5       public static void main(String[] args) {
6           int NoAlumno=2;
7           String DatoSimple;
8               String Datos[ ] = { "Matricula","Materia 1","Materia 2","Materia
            3",
                "Materia 4","Materia 5" };
9           int Registros [ ] [ ] = new int [ 30 ][ Datos.length ];
10
11          int Arreglo_Edad[]=new int[NoAlumno];
12              for (int i=0; i<NoAlumno;i++){
13                  for (int j=0; j<Datos.length; j++) {
14                      DatoSimple=JOptionPane.showInputDialog("Ingresa la "
                            + Datos[ j ] + " del alumno"+ (i+1));
```

```
15                          Registros[i][j]=Integer.parseInt(DatoSimple);
16                      }
17                  }
18
19              JOptionPane.showMessageDialog(null, "Datos capturados
                    completamente...");
20
21          }
22
23      }
```

Note cómo las asignaciones, y de un arreglo, son las mismas empleadas en un arreglo de una dimensión, lo que se debe tener presente es que ahora se maneja un arreglo con dos dimensiones, igual a una tabla, y para hacer referencia a una posición, tenemos que valernos de dos índices (Fila, columna).

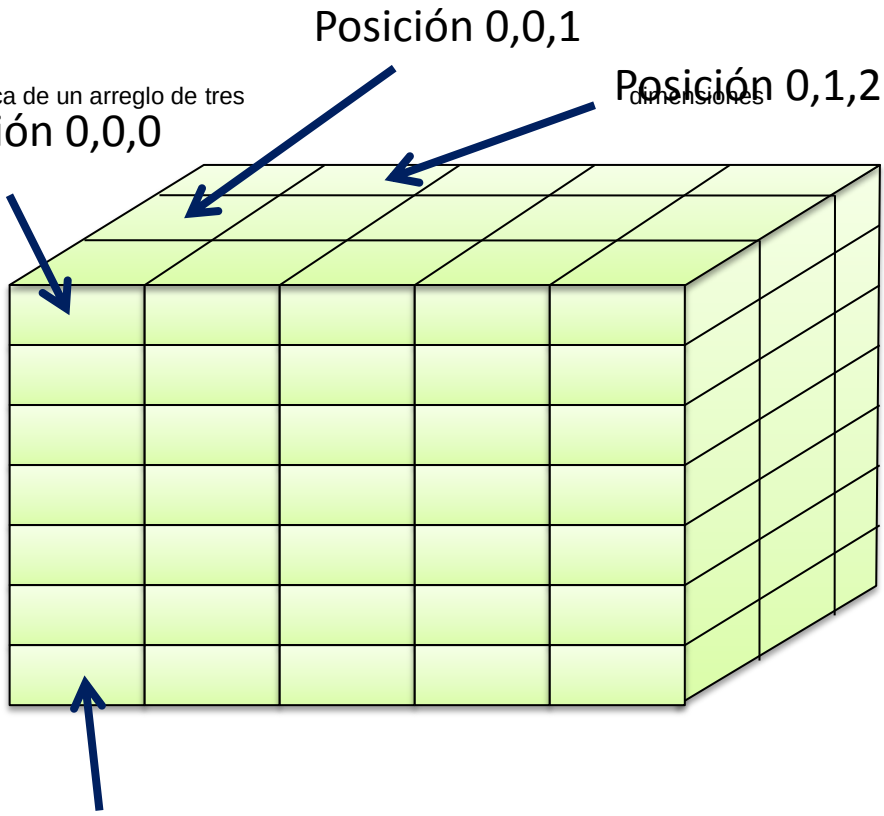
ACTIVIDAD DE APRENDIZAJE

1. Desarrolla un programa que permita la captura de las ventas de 30 Comisionistas de forma mensual, mostrando al final la venta Promedio, la venta más alta y más baja.

1.4. ARREGLOS DE TRES O MÁS DIMENSIONES

Esta estructura de datos está compuesta por *n* dimensiones, especificada en índices, y para hacer referencia a cada componente del arreglo es necesario utilizar *n* índices por cada dimensión empleada.

Fig. 1.5. Representación grafica de un arreglo de tres dimensiones



Para conocer el número total de elementos en un tipo de arreglos multidimensionales se emplea la siguiente fórmula:

No. TOTAL DE ELEMENTOS = D1 * D2* D3 * ...* Dn

donde:

D1, D2, D3…Dn= No. total de dimensiones

El anterior gráfico representa un arreglo de tres dimensiones con un total de 105 elementos, esto es porque tiene siete filas, cinco columnas y tres profundidades.

La sintaxis para la creación de un arreglo de tres dimensiones quedaría de la siguiente forma:

```
Tipo Nombre_Arreglo [ ] [ ] [ ]= new  Tipo [Fila] [ Columna ] [ Profundidad ];
```

Para comprender más claro los arreglos de tres dimensiones o más, realizaremos el siguiente problema:

En una empresa desean registrar las ventas de dos empleados que venden cinco productos, además de controlarlo por las tres semanas que estuvieron trabajando. La solución sería manejar un arreglo tridimensional de la siguiente forma:

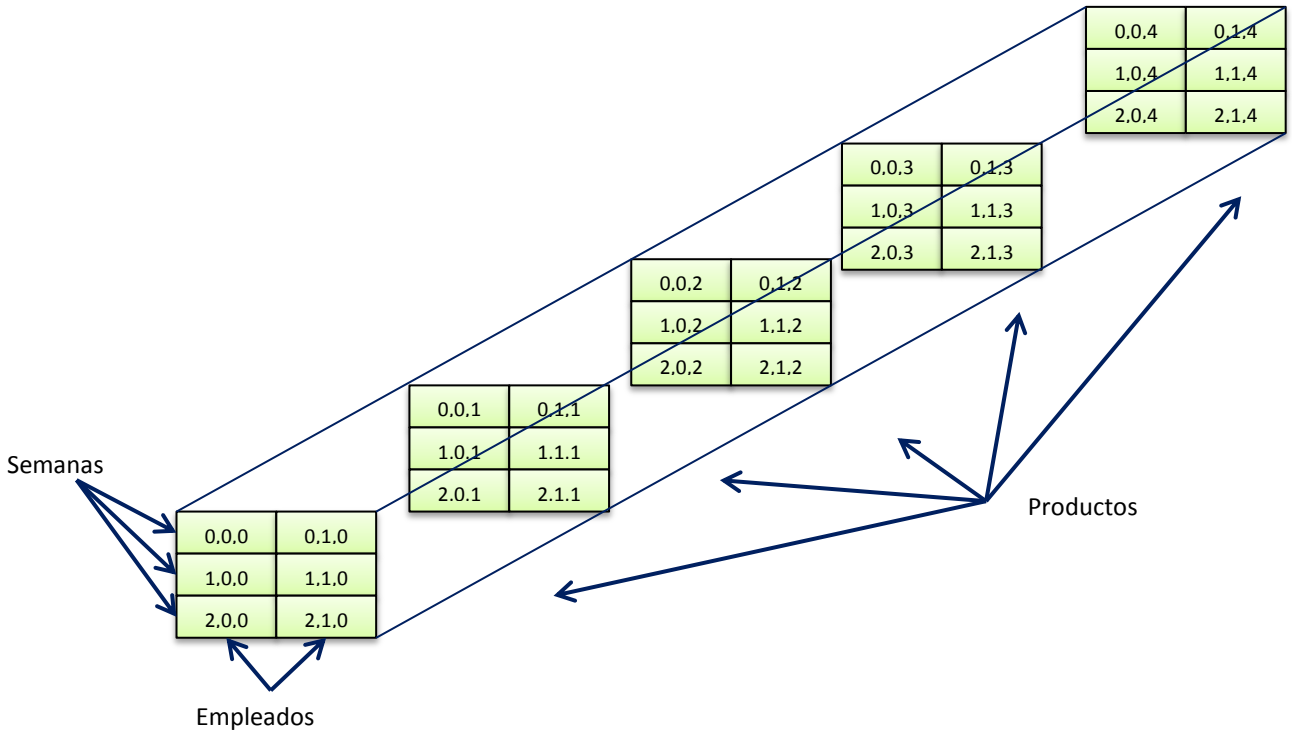


Fig. 1.6.- Arreglo de 3 filas (Semanas), 2 columnas (Empleados) y 5 profundidades (Productos)

El código quedaría expresado de la forma:

```
1       import javax.swing.JOptionPane;
2
3       public class tridimensional {
4               public static void main(String[] args) {
5                   String Valor;
6                   int Arreglo_Datos[][][]=new int[3][2][5];
7
8                   for (int i=0;i<3; i++){
9                      for (int j=0; j<2; j++){
10                        for (int k=0; k<5; k++){
11                              Valor=JOptionPane.showInputDialog("Semana:"+
(i+1)+" del
                                    vendedor: "+(j+1)+" del producto: "+(k+1));
12                              Arreglo_Datos[i][j][k]=Integer.parseInt(Valor);
13                        }
14                     }
```

```
15              }
16          }
17      }
```

ACTIVIDAD DE APRENDIZAJE

1.- Desarrolla un programa que permita la captura de las ventas de 20 Comisionista de forma mensual y de cinco productos diferentes, mostrando al final la venta Promedio, la venta más alta y más baja de cada vendedor.

AUTOEVALUACIÓN

1.- Puede definirse como un grupo o una colección finita, homogénea y ordenada de elementos. ( ) int x [ ] [ ]= int [3][4];

2.- En empleado para definir la posición de un dato en el arreglo ( ) int x [ ] = int [5];

3.- A los arreglos de dos dimensiones se le llaman ( ) Ordenado

4.- A los arreglos de una dimensión se le llaman ( ) Arreglo

5.- Ejemplo de un arreglo bidimensional ( ) Homogéneo

6.- Ejemplo de un arreglo unidimensional ( ) Finita

7.- Ejemplo de un arreglo tridimensional ( ) Índice

8.- Define que un arreglo tiene un límite de elementos ( ) int x [ ] [ ] [ ] = int [5][2][3];

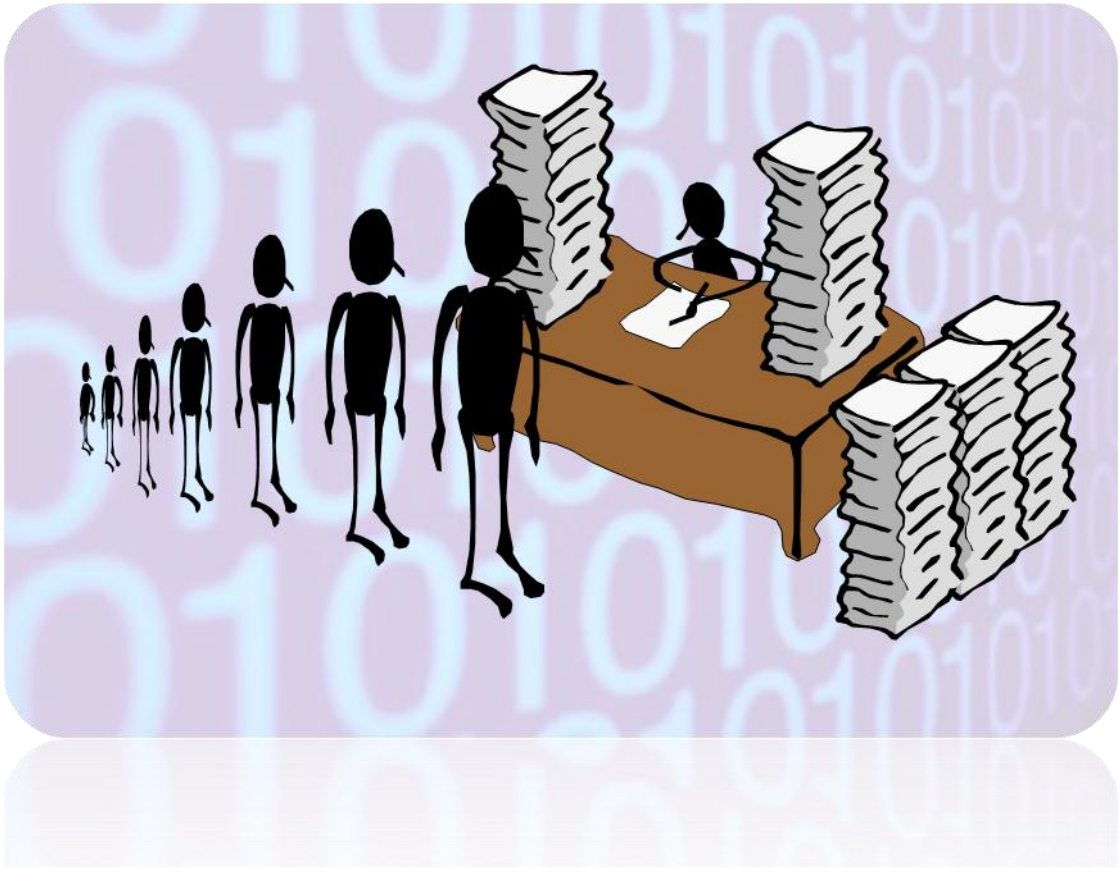
9.- Significa que un arreglo debe ser del mismo tipo ( ) Bidimensionales

10.- Define que cada elemento tiene su posición ( ) Unidimensionales

Respuesta: 6, 5, 10, 1, 9, 8, 2, 7, 3, 4.

UNIDAD 2

PILAS Y COLAS



OBJETIVO

Comprender los conceptos de *Pilas* y *Colas*, analizando las formas de manejar el acceso a datos, partiendo de sus fundamentos básicos.

TEMARIO

2.1. DEFINICIONES Y REPRESENTACIONES

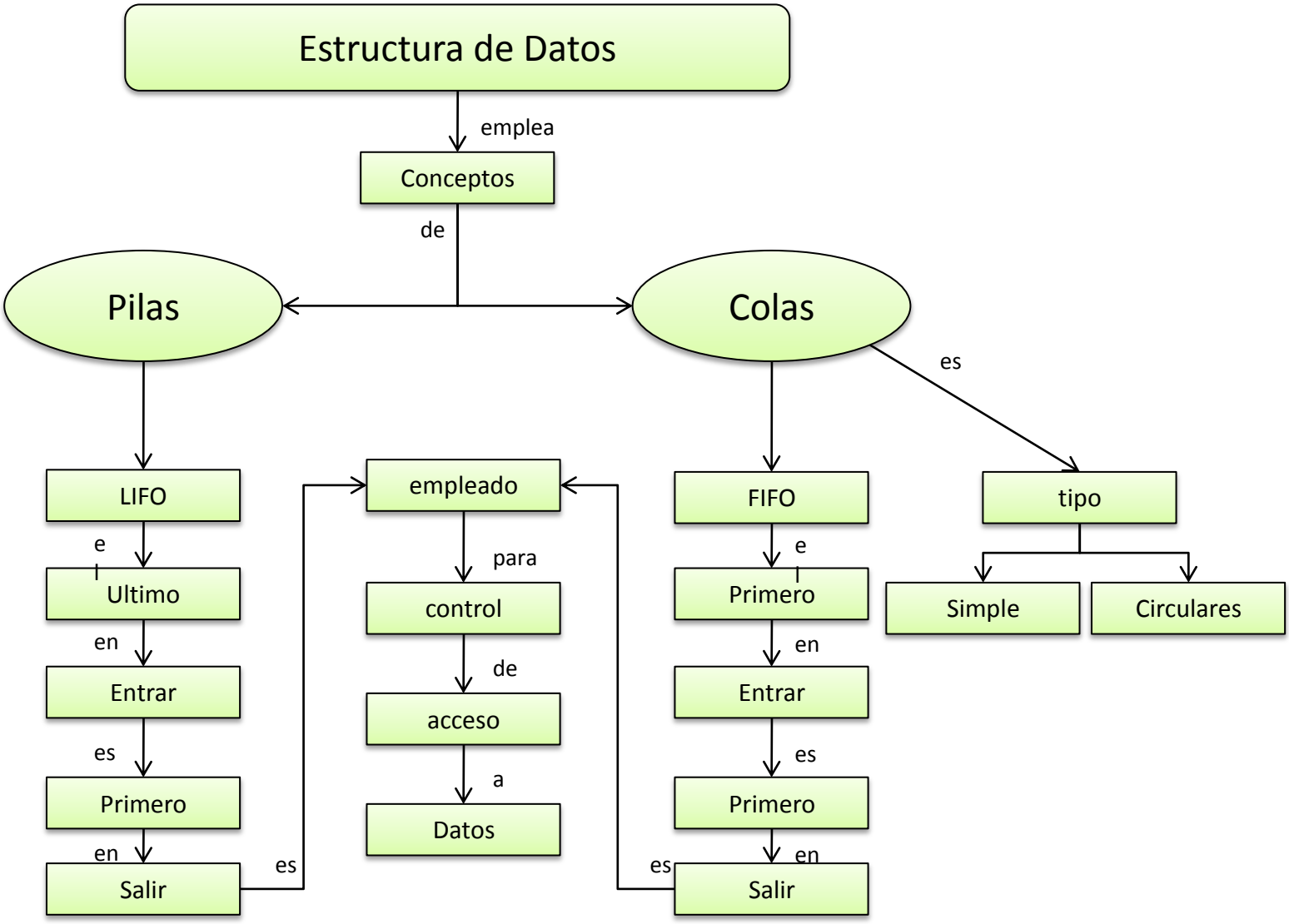
2.2. NOTACIONES INFIJAS, PREFIJAS, POSTFIJAS EN EXPRESIONES

2.3. INSERCIÓN Y REMOCIÓN DE DATOS EN UNA PILA (LIFO)

2.4. INSERCIÓN Y REMOCIÓN DE DATOS EN UNA COLA SIMPLE Y CIRCULAR

2.5. PROBLEMAS

MAPA CONCEPTUAL



INTRODUCCIÓN

Una de las características de los arreglos vistos en la unidad anterior, es que los datos pueden ser insertado o actualizados en cualquier posición y en cualquier momento. Sin embargo, dependiendo de las necesidades, en ocasiones necesitamos que cumplan con ciertas restricciones para controlar el acceso a los datos o bien para realizar determinados procesos en los cuales será necesario tener un orden.

En esta Unidad se comienza a explorar otro concepto fundamental en la *estructura de datos*, las Pilas y las Colas, a continuación se abordarán los conceptos, la implementación en los lenguajes de programación y su utilidad.

El manejo de las Pilas y las Colas incluye muchas restricciones a la forma de acceder a los datos, cada una de modo diferente, las cuales también se detallarán en el transcurso de la presente Unidad.

2.1. DEFINICIONES Y REPRESENTACIONES

Uno de los conceptos que más se emplean en las estructuras de datos lineales, en la elaboración de programas, son las *pilas*. Éstas son aplicadas en cuanto a las restricciones sobre el acceso a los datos del arreglo, ya sea para insertar o eliminar elementos, actualizando el contenido de los registros.

Iniciemos por definir el término fundamental de una pila como "una lista de elementos en la cual se puede insertar y eliminar elementos sólo por uno de los dos extremos."[3]

Para tener una comprensión más clara de lo que es una pila, imagine el acomodo de latas de un producto X en un centro comercial. O bien el apilamiento de libros en una biblioteca.

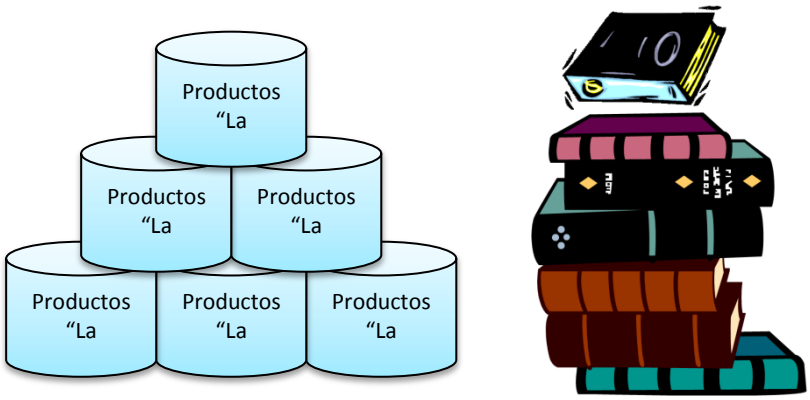


Fig. 2.1. Productos en un supermercado y libros apilados en la biblioteca

Según las imágenes anteriores, podemos deducir el razonamiento de cómo extraer un elemento de la pila, los componentes de una pila serán empleados en orden inverso al que se colocaron. Es decir, el último en entrar debe ser el primero en salir.

Las pilas son estructuras utilizadas muy a menudo como herramientas de programación de tipo LIFO (Last in-First out), ya que permiten el acceso solo a un elemento a la vez: el último elemento insertado. La mayoría de los procesadores utilizan una arquitectura basada en pilas.

La pila se considera un grupo ordenado de elementos, teniendo en cuenta que el orden de los mismos depende del tiempo que lleven "dentro" de la estructura. Las pilas son empleadas en el desarrollo de sistemas informáticos y software en general. Por ejemplo, el sistema de soporte en

[3] Osvaldo Cairó/Silvia Guardati, *Estructuras de datos*, p. 75. *Cfr*. http://www.cesarportela.com.ar/facultad/08-grafos2005.pdf

tiempo de compilación y ejecución de Pascal, utiliza una pila para llevar la cuenta de los parámetros de procedimientos y funciones, variables locales, globales y dinámicas. Este tipo de estructuras también son utilizadas para traducir expresiones aritméticas o cuando se quiere recordar una secuencia de acciones u objetos en el orden inverso del ocurrido.

Ahora conoceremos el concepto de Cola y se definirá como "una estructura de almacenamiento donde los datos van a ser insertados por un extremo y serán extraídos por otro".[4] El concepto anterior se puede simplificar como FIFO (first-in, first-out), esto es, el primer elemento en entrar debe ser el primero en salir.

Los ejemplos sobre este mecanismo de acceso, son visibles en las filas de un banco, los clientes llegan (entran) se colocan en la fila y esperan su turno para salir.

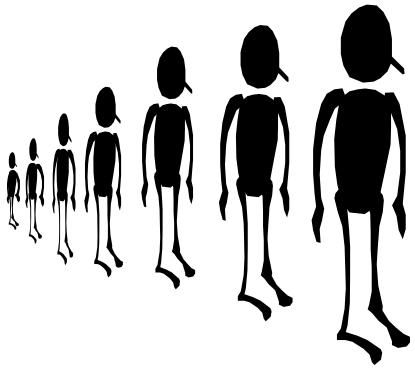


Fig. 2.2.- Ejemplo grafico de las colas en un banco.

Varios hechos de la vida que podemos aplicar en el concepto de pilas son, por ejemplo, cuando las personas hacen fila para esperar el uso de un teléfono público, para subir al transporte escolar o un autobús, para realizar los pagos en un supermercado, etc.

ACTIVIDAD DE APRENDIZAJE

1.-Realizar una investigación y entregar un resumen donde se comprenda el concepto pilas y colas.

[4] Goodrich/Tamassia, *Estructura de datos y algoritmos en Java*, p. 215.

2.2. NOTACIONES INFIJAS, PREFIJAS, POSTFIJAS EN EXPRESIONES

La implementación de las pilas en una aplicación para la resolución de problemas, es común; esto se debe a que dentro de las estructuras de datos, el uso de las pilas permite mejorar la forma de analizar y resolver problemas matemáticos.

Para tomar problemas reales y aclarar el uso de las pilas, se empleará el concepto de las notaciones, las cuales se pueden clasificar en:

Notaciones infijas: Son llamadas así a la anotación de fórmulas matemáticas que emplean operadores, por ejemplo 2+8, es decir, es la forma de escribir nuestras operaciones de uso común.

Notaciones prefijas: Estas permiten expresar nuestras operaciones matemáticas, colocando los operadores al inicio de la expresión, continuando con el ejemplo anterior quedaría + 28.

Notación posfija: Permite escribir las operaciones colocando los operadores al final de la expresión, por ejemplo: 28+.

Véase la siguiente expresión algorítmica para la conversión de una expresión infija en postfija (RPN - Reverse Polish Notation).

1. Aumentar la pila
2. Inicializar el conjunto de operaciones
3. Mientras no exista error y no es fin de la expresión infija realizar:
    - Si el carácter es igual a
        1. PARENTESIS IZQUIERDO. Colocarlo en la pila
        2. PARENTESIS DERECHO. Extraer y desplegar los valores hasta encontrar paréntesis izquierdo. Pero NO desplegarlo.
        3. UN OPERADOR.
            - Si la pila está vacía o el carácter tiene más alta prioridad que el elemento del tope de la pila insertar el carácter en la pila.
            - En caso contrario extraer y desplegar el elemento del tope de la pila y repetir la comparación con el nuevo tope.
        4. OPERANDO. Desplegarlo.
4. Extraer y mostrar los elementos de la pila hasta que se agote.

Ejemplo de conversión de expresiones fijas a postfija

Expresión en formato fija	Expresión en formato RPN
8+2	2 8 +
7	7
5+2/6	2 6 / 5 +
8+ 5/6 - 2/x	85 6 / + 2 x / -

Algoritmo para evaluar una expresión RPN

1. Incrementar la pila
2. Repetir
    - Tomar un caracter.
    - Si el caracter es un operando colocarlo en la pila.
    - Si el caracter es un operador entonces tomar los dos valores del tope de la pila, aplicar el operador y colocar el resultado en el nuevo tope de la pila. (Se produce un error en caso de no tener los 2 valores)
3. Hasta el fin de la expresión RPN.

ACTIVIDAD DE APRENDIZAJE

1.- Realizar una investigación y entregar un resumen, donde se comprenda el concepto de las notaciones.

2.3. INSERCIÓN Y REMOCIÓN DE DATOS EN UNA PILA (LIFO)

Las pilas no son estructuras de datos fundamentales, esto es, no se encuentran definidas como tales en los lenguajes de programación. Las pilas pueden representarse mediante lo siguiente:

- Arreglos.
- Listas enlazadas.

Como en la primera Unidad se estudió sobre los arreglos, sea más conveniente su implementación para la creación de las pilas y colas. Por lo tanto, se debe definir el tamaño máximo de la pila, además de un apuntador al

último elemento insertado en la pila, el cual se denomina SP. La representación gráfica de una pila es la siguiente, denotando como TOPE al T-enésimo elemento de la pila que se inserta:

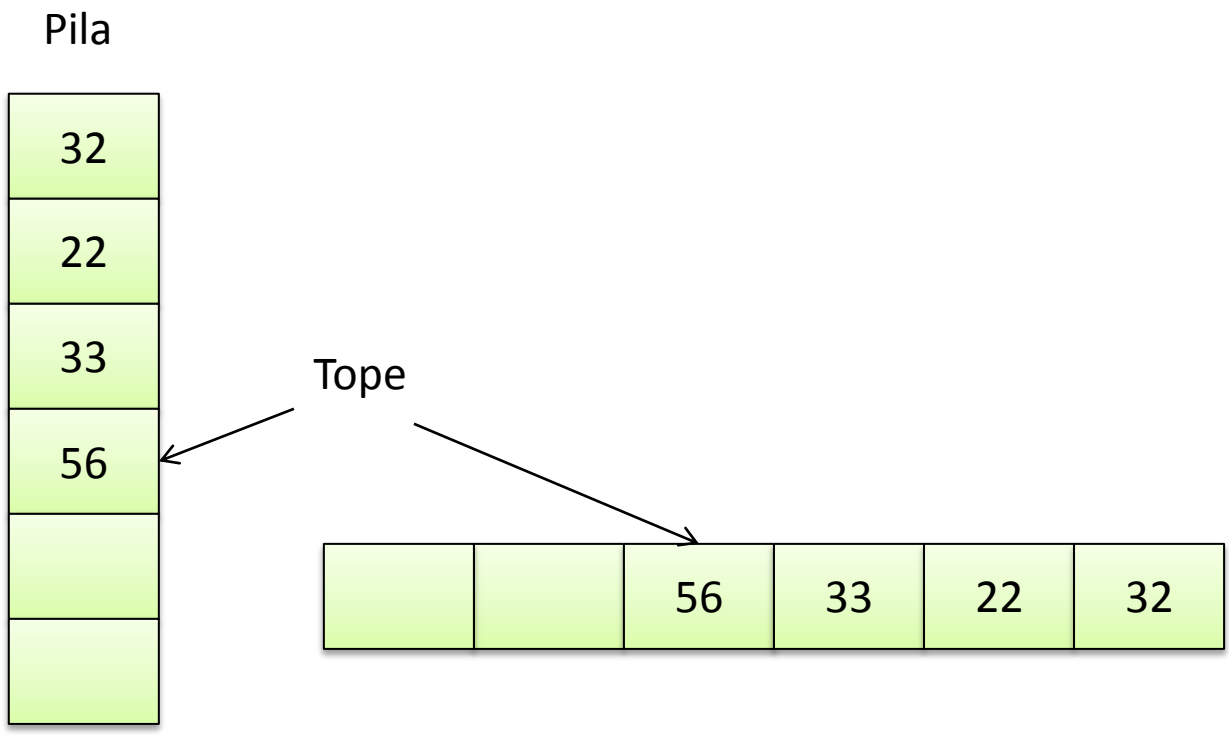


Fig. 2.2.- Representación de una pila

Para implementar el concepto de pilas, se usará como base a los arreglos, éstos estarán limitados a un número máximo de elementos, lo que no permitirá agregar otros registros.

Existen cuatro operaciones básicas que son válidas para el tipo de datos pila, las cuales son las siguientes:

- Lleno (Pila).
- Vacio (Pila).
- Agregar (Elementos, Pila).
- Eliminar (Pila).

Los algoritmos principales serán Agregar y Eliminar, debido a que Crear y Lleno sólo devolverán valores boleanos, para saber si una pila está creada o si tiene datos.

Pseudocódigo 1.- Vacía (Pila, Tope, Res1)

// Este algoritmo verifica si la pila tiene datos, asignando a Res1 el valor de verdadero

```
Si Tope=0 entonces
        Res1 ← Verdadero
Si_no
        Res1 ←Falso
Fin_Si
```

Pseudocódigo 2.- Lleno (Pila, Tope, Max, Res2)

// Este algoritmo verifica si la pila tiene espacio, asignando a Res2 el valor de
// verdadero, definiendo como Max al tamaño del arreglo

```
Si Tope=Max  entonces
        Res2 ← Verdadero
Si_no
        Res2←Falso
Fin_Si
```

Pseudocódigo 3.- Agregar

// Este algoritmo añade un elemento a la pila

```
Llamar Lleno (Pila, Tope, Max, Res)
Si Resp2=Verdadero entonces
        Mostrar “Error pila llena, desbordamiento de datos”
Si_no
        Tope← Tope + 1
        Pila[Tope] ←Dato
Fin_Si
```

Pseudocódigo 4.- Eliminar

// Este algoritmo añade un elemento a la pila

```
Llamar Vacio(Pila, Tope,  Res)
Si Resp1=Verdadero entonces
        Mostrar “Error pila vacía”
Si_no
```

Tope← Tope - 1

Dato ← Pila[Tope]

Fin_Si

ACTIVIDAD DE APRENDIZAJE

1. Desarrolla un programa que permita controlar los autos que entran en un estacionamiento, empleando el número de placa, a través de pilas.

2.4. INSERCIÓN Y REMOCIÓN DE DATOS EN UNA COLA SIMPLE Y CIRCULAR

Podemos representar el uso de las colas de dos formas:

- Como arreglos.
- Como listas ordenadas.

Al igual que las pilas, trataremos las colas como arreglos, en donde debemos definir el tamaño de la cola y dos apuntadores, uno para accesar el primer elemento de la lista y otro que guarde el último. En lo sucesivo, al apuntador del primer elemento se le denominará Inicial, al de el último elemento Final y Max, para definir el número máximo de elementos en la cola.

La cola lineal es un tipo de almacenamiento creado por el usuario que trabaja bajo la técnica FIFO (primero en entrar primero en salir). Las colas lineales se representan gráficamente de la siguiente manera:

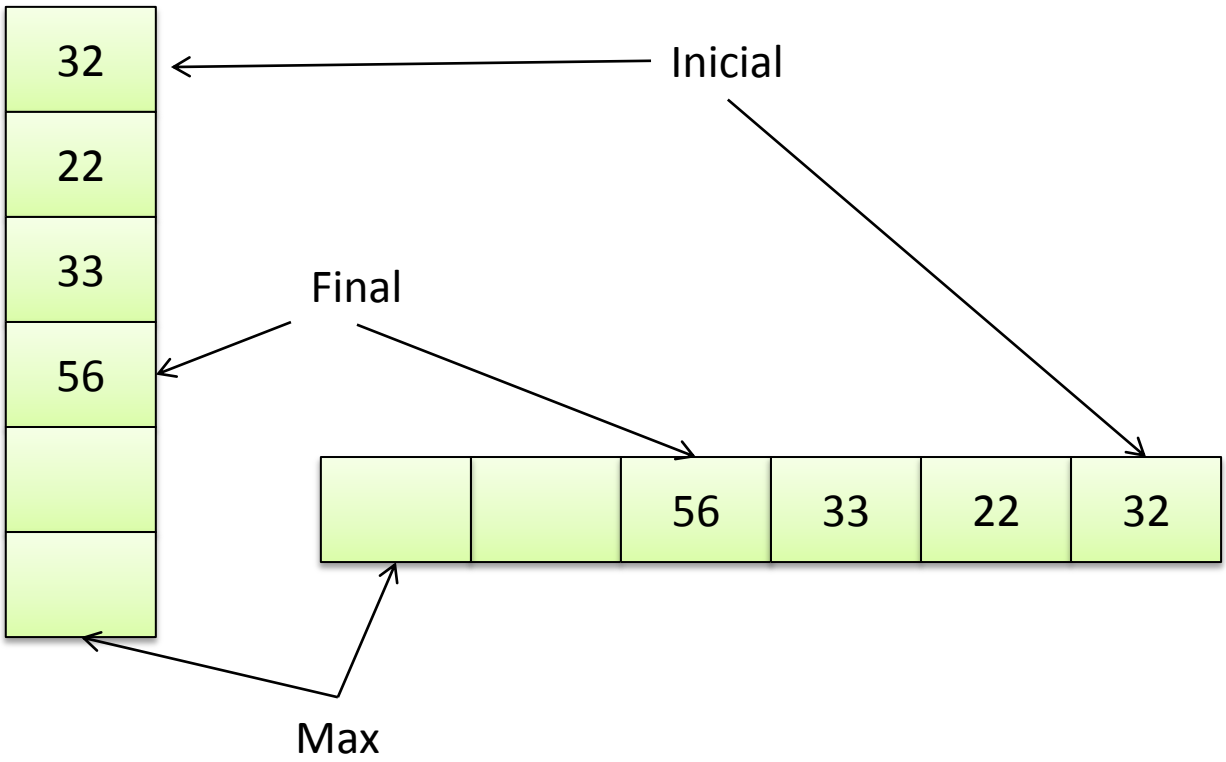


Fig. 2.3. Representación gráfica de una Cola

Las colas también se utilizan en muchas maneras en los sistemas operativos para planificar el uso de los distintos recursos de la computadora. Uno de estos recursos es la propia CPU (Unidad Central de Procesamiento).

Si está trabajando en un sistema multiusuario, cuando le dice a la computadora que ejecute un programa concreto, el sistema operativo añade su petición a su “cola de trabajo”.

Cuando su petición llega al frente de la cola, el programa solicitado pasa a ejecutarse. Igualmente, las colas se utilizan para asignar tiempo a los distintos usuarios de los dispositivos de entrada/salida (E/S), impresoras, discos, cintas y demás. El sistema operativo mantiene colas para peticiones de imprimir, leer o escribir en cada uno de estos dispositivos.

Las operaciones que se pueden implementar en una cola son las siguientes:

- Insertar.
- Eliminar.

Hay que recordar siempre que la inserción se realiza al Final de la cola, mientras que en Inicial llevará el control de las eliminaciones.

```
Algoritmos 1. InsertarDato
Si Final < Max entonces
        Final ← Final + 1
        Cola [ Final ] ←Dato
        Si Final=1 entonces
                Inicial ← 1
        Si_no
                Mostrar “Error”
        Fin_si
Fin_si
```

```
Algoritmo 2.- EliminarDato
Si Inicial ≠ 0 entonces
        Datos ←Cola [ Inicial ]
        Si Inicial= Final entonces
                Frente←0
                Final ← 0
        Si_no
                Frente ← Frente – 1
        Fin_si
Fin_Si
```

Colas circulares

Las colas lineales tienen un grave problema, como las extracciones sólo pueden realizarse por un extremo, puede llegar un momento en que el apuntador Inicial sea igual al máximo número de elementos en la cola, siendo que al frente de la misma existan lugares vacíos, y al insertar un nuevo elemento nos mandará un error.

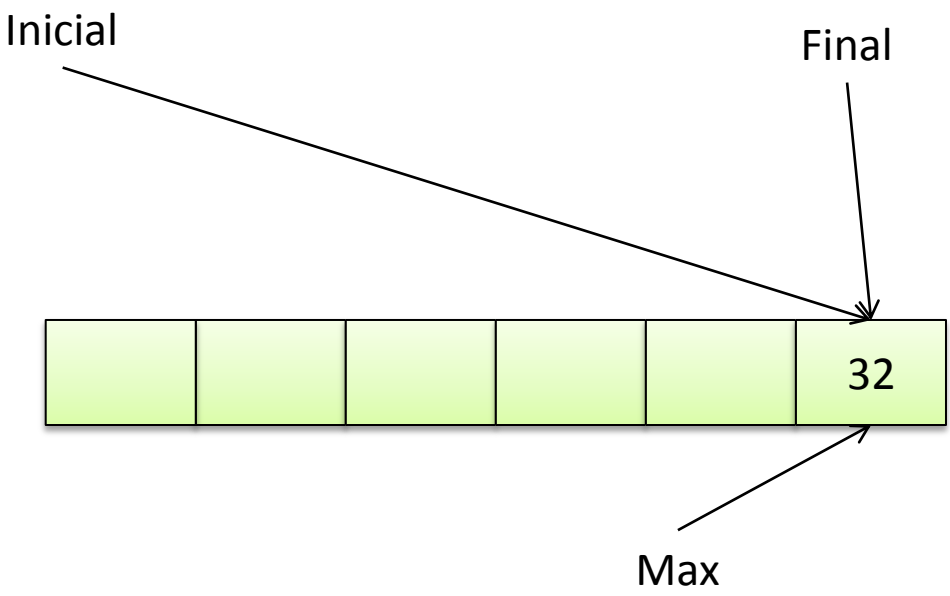


Fig. 2.4.- Problemas con las Colas al llegar al elemento final

Para solucionar el problema de desperdicio de memoria, se implementaron las colas circulares, en las cuales existe un apuntador desde el último elemento al primero de la cola. Así, al momento de no haber más posiciones al final de la cola, se reasignan a los elementos vacios de la cola.

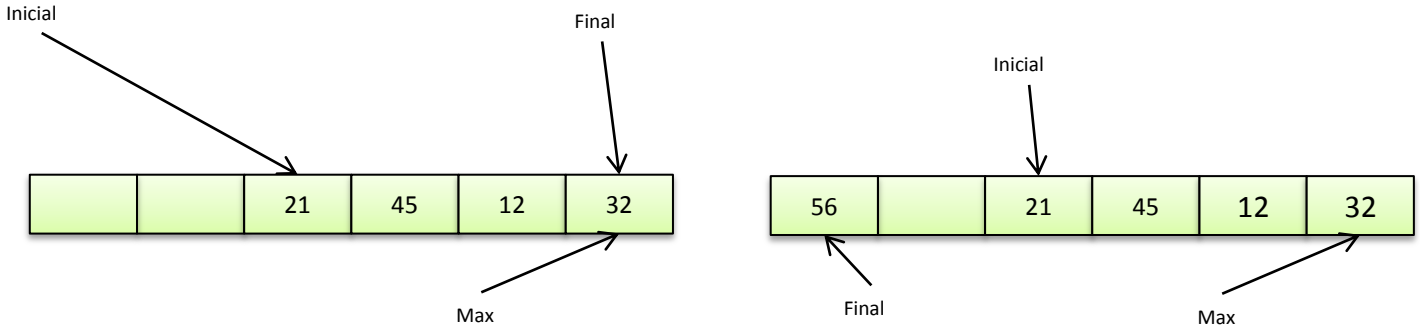


Fig. 2.5. Cola circular

Algoritmo para InsertarColaCircular

Si ((Final = Max) y (Frente = 1)) o ((Final + 1) = Frente) entonces

    Mostrar “Desbordamiento, cola llena”

Si_no

    Si Final=Max

        Final ←1

    Si_no

```
            Final ← Final +1
        Fin_Si
        ColaCir[Final]←Datos
        Si Frente=0 entonces
            Frente←1
        Fin_Si
Fin_Si

Algoritmo para EliminarColaCircular
Si Frente=0
        Mostrar “Error cola vacia”
Si_no
        Dato=ColaCir[Frente]
        Si Frente=Final entonces
            Frente ←0
            Fina←0
        Si_no
            Si Frente=Max entonces
                Frente ←1
            Si_no
                Frente←Frente +1
            Fin_si
        Fin_si
Fin_Si
```

ACTIVIDAD DE APRENDIZAJE

1. Desarrolla un programa que permita controlar los autos que entran en un estacionamiento, empleando el número de placa, a través de colas circulares.

2.5 PROBLEMAS

A continuación, se realizarán dos programas que permitirán observar la codificación en el lenguaje de programación Java, sobre el manejo de las Pilas y Colas, los métodos de acceso y el cumplimiento de sus algoritmos.

Problema 1. Empleando las estructuras de datos, analizar el siguiente programa que permite la captura y eliminación de códigos de libros, en un arreglo de 10 elementos controlando los accesos través de pilas.

```
import javax.swing.JOptionPane;

public class Libros {
   int Max=10, Tope=0;
   int Pila[]= new int[Max];
   String Dato;
   boolean Res1=false, Res2=false;
   public static void main(String arg[]){
      Libros Acceso=new Libros();
   }

   public Libros(){
      int op=0;
     while(op!=4){
      op=Integer.parseInt(JOptionPane.showInputDialog(
            "Opciones Para La Pila" +
            "\n [1] Agregar Dato" +
            "\n [2] Eliminar Dato" +
            "\n [3] Mostrar Pila" +
            "\n [4] Terminar"
            ));
      switch(op){
         case 1:{AgregarPila(); break;}
         case 2:{BorrarPila(); break;}
         case 3:{MostrarPila(); break;}
      }

     }
   }
```

```
public void AgregarPila(){
          PilaLleno();
          if (Res1==true){
             JOptionPane.showMessageDialog(null, "Pila  Llena");
          } else{
             Dato=JOptionPane.showInputDialog("Codigo:");
             Pila[Tope]=Integer.parseInt(Dato);
             Tope++;
          }
}

public void PilaLleno(){
        if (Tope==Max) Res1=true;
        else   Res1=false;
}

public void MostrarPila(){
    String Cadena="";
     for (int i=0;i<Max;i++){
         Cadena+="Posicion "+ i +" ---> "+Pila[i]+"\n";
     }
     JOptionPane.showMessageDialog(null, Cadena);
}

public void BorrarPila(){
    PilaVacia();
    if (Res2==true){
       JOptionPane.showMessageDialog(null, "Pila  Vacia");
    } else{
       Tope-=1;
       JOptionPane.showMessageDialog(null, "Dato eliminado "+Pila[Tope]);
       Pila[Tope]=0;

    }
```

```
    }

    public void PilaVacia(){
            if (Tope==0) Res2=true;
            else Res2=false;
    }

}
```

Problema 2. Empleando las estructuras de datos, analizar un programa que permita la captura de una lista de clientes mediante su código, para que sean atendidos por un personal, dicha lista será manejada en un arreglo de 20 elementos, permitiendo ser controlado por el criterio de las Colas (Primero en llegar, primero en salir).

```
import javax.swing.JOptionPane;

public class Clientes {
    int Max=20, Frente=-1, Final=0;
    int Cola[]= new int[Max];
    String Dato;
    boolean Res1=false, Res2=false;

    public static void main(String arg[]){
            Clientes Acceso=new Clientes();
    }

    public Clientes(){
    int op=0;
      while(op!=4){
        op=Integer.parseInt(JOptionPane.showInputDialog(
                "Opciones el registro de clientes" +
                "\n [1] Agregar Dato" +
```

```
            "\n [2] Eliminar Dato" +
            "\n [3] Mostrar Elementos" +
            "\n [4] Terminar"
            ));
      switch(op){
         case 1:{AgregarCola(); break;}
         case 2:{BorrarCola(); break;}
         case 3:{MostrarCola(); break;}
      }

     }
   }

   public void MostrarCola(){
       String Cadena="";
        for (int i=0;i<Max;i++){
            Cadena+="Posicion "+ i +" ---> "+Cola[i]+"\n";
        }
        JOptionPane.showMessageDialog(null, Cadena);
   }

   public void AgregarCola(){
          if (Final<Max){
              Dato=JOptionPane.showInputDialog("Codigo:");
              Cola[Final]=Integer.parseInt(Dato);
              Final++;
              if (Final==1) Frente=0;
          } else
                JOptionPane.showMessageDialog(null,"No hay elementos
disponibles en Cola..");

   }

   public void BorrarCola(){
```

```
        if(Frente!=-1){
            JOptionPane.showMessageDialog(null, "Dato eliminado "+Cola[Frente]);
            Cola[Frente]=0;
            if (Frente==(Final-1)){
                System.out.print(Frente+"-"+Final);
                    Frente=-1;
                    Final=0;
            } else {
                Frente++;
            }
        }
        else {
            JOptionPane.showMessageDialog(null,"Datos Vacios");
        }
    }

}
```

AUTOEVALUACIÓN

1.- Es una lista de elementos en la cual se puede insertar y eliminar elementos sólo por uno de los extremos. ( ) Tope

2.- Estos elementos son eliminados en orden inverso al que se insertaron. Es decir, el último en entrar es el primero en salir. ( ) Agregar

3.- Es una estructura de almacenamiento donde los datos van a ser insertados por un extremo y serán extraídos por otro ( ) Pilas

4.- Es un método empleado en las filas de un banco, los clientes llegan se colocan en la fila y ( ) Arreglos

esperan su turno para salir.

5.- Es una manera de representar las pilas ( ) Colas

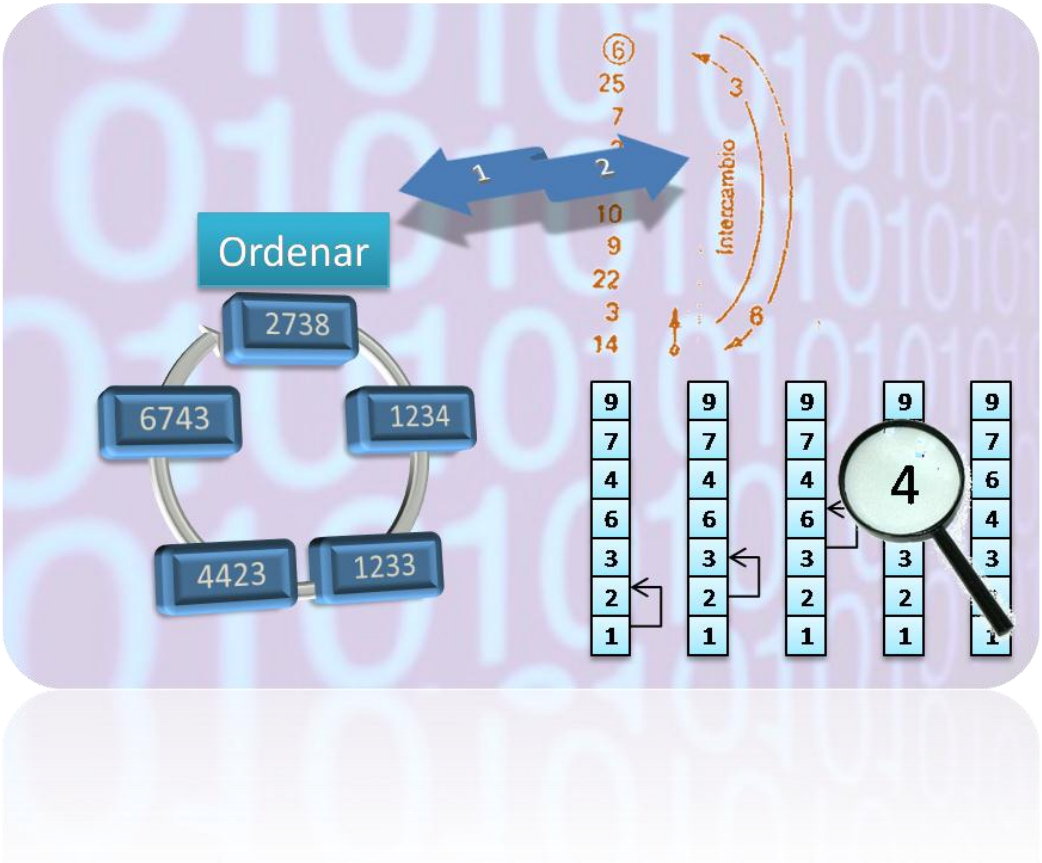
6.- Es el T-enésimo elemento de la pila que se inserta ( ) Colas

7.- Es una operación básica de las pilas ( ) Pilas

Respuestas: 6, 7, 2, 5, 3, 4, 1

UNIDAD 3

ALGORITMOS DE ORDENAMIENTO Y BÚSQUEDA

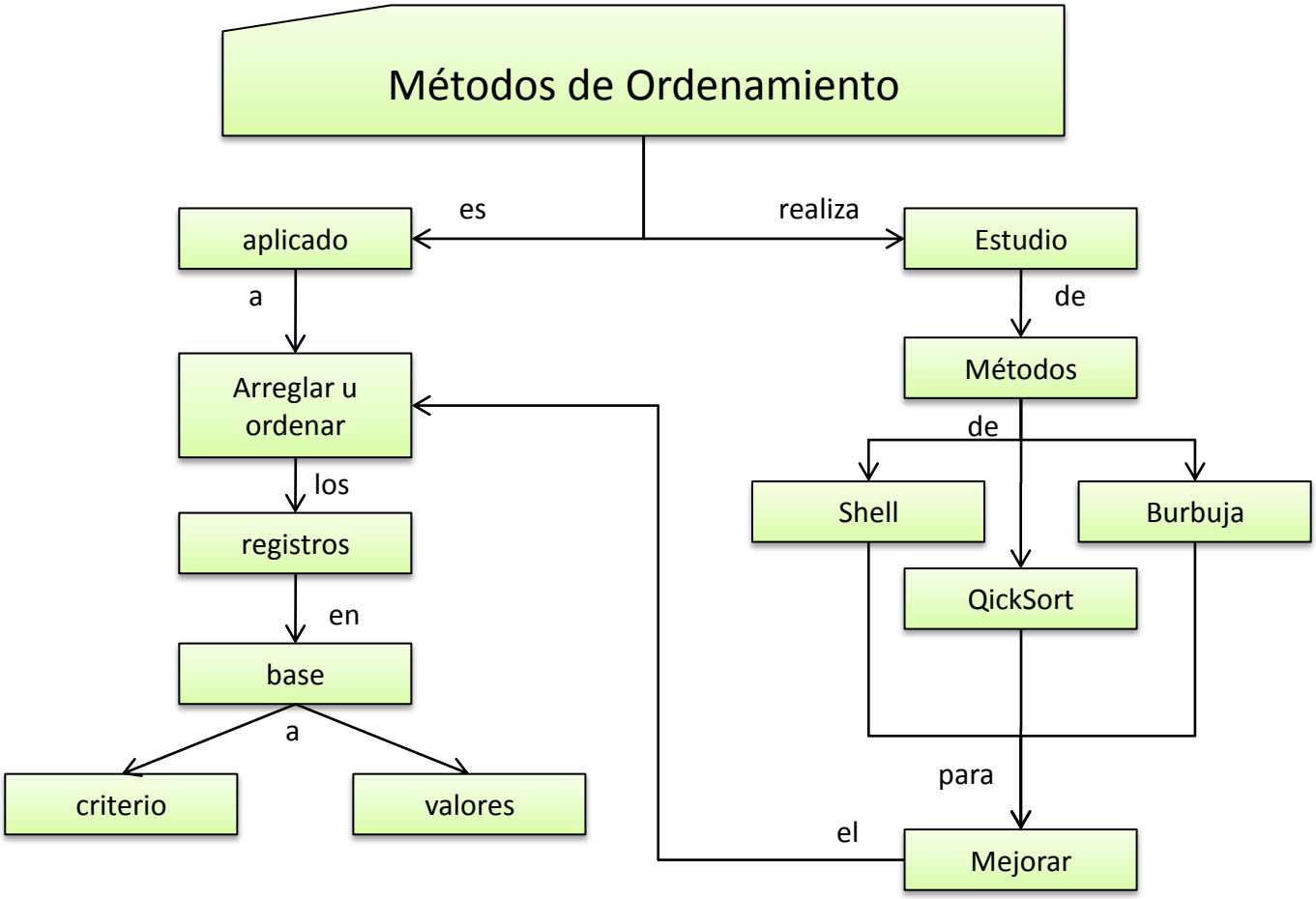


OBJETIVO

Identificar y analizar los métodos más comunes para el ordenamiento y las búsquedas de la información dentro de estructuras de datos.

TEMARIO

MAPA CONCEPTUAL



INTRODUCCIÓN

En la siguiente Unidad emplearemos un pequeño descanso sobre nuevos conocimientos en estructura de datos, en su lugar, profundizaremos sobre mecanismos que nos permitirán mejorar la calidad de la información contenida en los arreglos y nos referimos a los métodos de ordenamiento y búsqueda.

La búsqueda y ordenamiento son procesos estrechamente relacionados, la búsqueda es el proceso de localizar un registro con valor particular, mientras que el ordenamiento es el proceso de arreglar los registros de tal manera que queden ordenados en base a un criterio o valor.

En el procesamiento de datos podemos encontrar dos tipos de ordenamientos, los de arreglo y los de archivo. Los primeros se refieren a los datos que se encuentran almacenados en la memoria de la computadora como pueden ser los arreglos. Los de archivo son aquellos que se encuentran almacenados en un medio de almacenamiento como los discos duros.

Ahora nos enfocaremos en la implementación y prueba de los métodos de ordenamiento por arreglos, o como se conoce *métodos de ordenamiento interno*.

3.1. MÉTODO DE BURBUJA

Este método consiste en acomodar los vectores moviendo el mayor hasta la última casilla, comenzando desde la casilla cero, esto se logra comparando valores de llaves y al intercambiarlos si no están en una posición relativa correcta.

Este algoritmo es muy deficiente ya que al ir comparando las casillas para buscar el siguiente más grande, éste vuelve a comparar las ya ordenadas. A pesar de ser el algoritmo de ordenamiento más deficiente que hay, éste es el más usado en todos los lenguajes de programación.

Este método logra la idea básica de la burbuja, que cada valor flote a su posición adecuada mediante comparaciones en pares. Cada paso hace que el valor suba a su posición final, como una burbuja. Ilustremos el concepto con los siguientes valores:

4
7
3
2
9
1
6

Como se ha dicho, la burbuja sube, por lo que cada valor se compara con el que se encuentra arriba de ella, y se intercambia, si la de arriba es más pequeña, después de una pasada, habrá cambios en el ordenamiento como se ilustra a continuación:

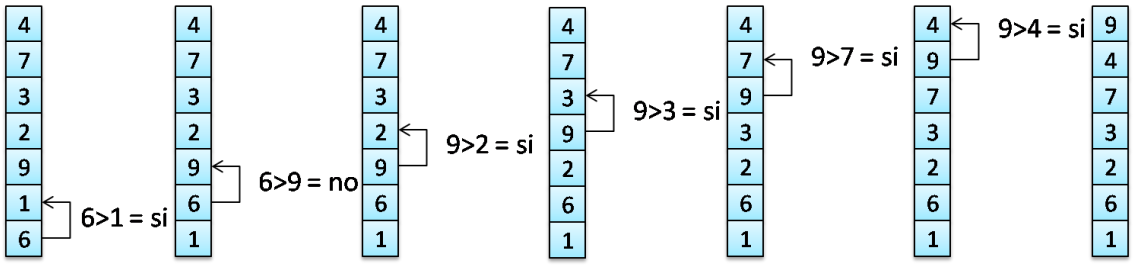


Fig. 3.1 Método de Burbuja Primer ordenamiento

Después del primer ordenamiento se obtiene el número más alto y reinician la comparación desde el último valor hasta completar un ciclo de *n* veces.

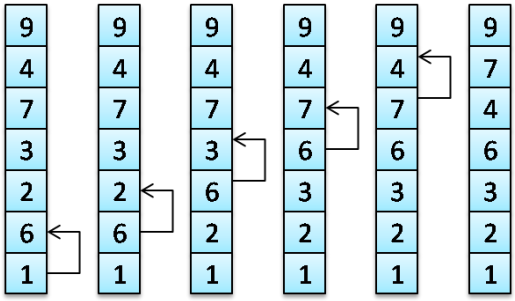


Fig. 3.2 Método de Burbuja segundo ordenamiento

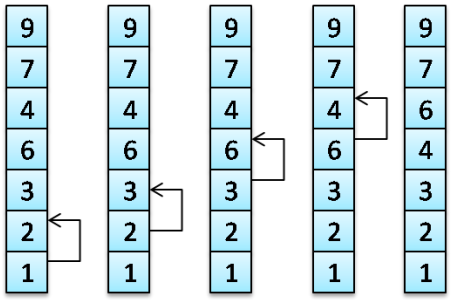


Fig. 3.3 Método de Burbuja tercer ordenamiento

Algoritmo:

```
Repetir con I desde 2 hasta N
        Repetir con J desde N hasta I
                Si  A [ J – 1 ] > A [ J ] entonces
                        Tempo ←A [ J-1 ]
                        A [ J-1 ] ← A [ I ]
                        A [ I ] ← Tempo
                Fin_Si
        Fin_Repetir
Fin_Repetir
```

ACTIVIDAD DE APRENDIZAJE

1. Desarrolla un programa que permita ingresar y ordenar la siguiente numeración 33, 45, 67, 8, 12, 32, 56, 7, 9, 10, 22, 45, 98, 9. De forma ascendente y descendente, usando el método de burbuja.

3.2. MÉTODO SHELL

Este método también se conoce con el nombre de inserción con incrementos decrecientes. El método de *shell* es una versión mejorada del método de inserción directa, recibe ese nombre en honor a su autor Donald L. Shell quien lo propuso en 1959.

En el método de ordenación Shell propone que las comparaciones entre elementos se efectúen con saltos de mayor tamaño pero con incrementos decrecientes, así los elementos quedarán ordenados en el arreglo.

Ejemplo: Se desean ordenar las siguientes claves del arreglo : 15, 67, 08, 16, 44, 27, 12, 35, 56, 21, 13, 28, 60, 36, 07, 10

Primera Pasada.- Los elementos se dividen en 8 grupos:
15, 67, 08, 16, 44, 27, 12, 35 | 56, 21, 13, 28, 60, 36, 07, 10

La ordenación produce:
15, 21, 08, 16, 44, 27, 07, 10, 56, 67, 13, 28, 60, 36, 12, 35

Segunda Pasada .- Se dividen los elementos en 4 grupos:
15, 21, 08, 16 | 44, 27, 07, 10 | 56, 67, 13, 28 | 60, 36, 12, 35

La ordenación produce:
15, 21, 07, 10, 44, 27, 08, 16, 56, 36, 12, 28, 60, 67, 13, 35

Tercera Pasada .- Se divide los elementos 2 grupos
15, 21 | 07, 10 | 44, 27 | 08, 16 | 56, 36 | 12, 28 | 60, 67 | 13, 35

La ordenación produce:
07, 10, 08, 16, 12, 21, 13, 27, 15, 28, 44, 35, 56, 36, 60, 67

Cuarta Pasada.- Divida los elementos en un solo grupo.

La ordenación produce:

07, 08, 10, 12, 13, 15, 16, 21, 27, 28, 35, 36, 44, 56, 60, 67

ACTIVIDAD DE APRENDIZAJE

1. Realizar y entregar una investigación sobre el método Shell con ejemplos
2. Desarrolla un programa que permita ingresar y ordenar la siguiente numeración 33, 45, 67, 8, 12, 32, 56, 7, 9, 10, 22, 45, 98, 9. De forma ascendente y descendente, usando el método de Shell.

3.3. MÉTODO DE QUICKSORT

El método Quickasort emplea la técnica de “divide y vencerás”, el cual se refiere a tomar el arreglo y empezar a dividirlos en secciones más pequeñas, para así poder empezar a ordenar esos arreglos.

El procedimiento consisten en tomar un valor que fungirá como punto de pivote, luego se ordenan los elementos, los menores se mueven a la izquierda y los mayores a la derecha, siempre partiendo del pivote. Continuando se realiza el mismo procedimiento a las otras partes de las divisiones del arreglo.

La mayoría de los programadores realizan los siguientes pasos: primero se toma como punto de pivote al primer elemento del arreglo. Luego, de izquierda a derecha, se busca el elemento mayor, así como de derecha a izquierda el elemento menor, una vez encontrados éstos, se intercambian de posición. Estos pasos se realizan una y otra vez hasta que los dos procesos se encuentren y no haya más elementos.

A continuación aplicaremos el método de ordenamiento QuickSort para ordenar el siguiente arreglo: {21,40,4,9,10,35}.

Como se explicó, primero se toma como punto de pivote al primer elemento, en este caso el 21. En segundo lugar se realizan las búsquedas (Izquierda-Derecha) se encuentra el valor 40, la otra búsqueda encuentra el valor 10 (Derecha-Izquierda). Se realiza el intercambio quedando de la siguiente forma:

{21,10,4,9,40,35}

El ciclo de las búsquedas se reinicia encontrando el valor 40 como mayor, y el valor 9 como el elemento menor, en este caso ya se cruzaron las búsquedas, así que se procede a cruzar el menor con el pivote, quedando:

{9,10,4,21,40,35}

Partiendo del punto del pivote, dividiremos el arreglo en otros dos más pequeños {9,10,4} y el {40,35}, ahora el método de ordenamiento QuickSort se aplica a cada uno de los arreglos.

ACTIVIDAD DE APRENDIZAJE

1. Realizar y entregar una investigación sobre el método Quicksort con ejemplos.
2. Desarrolla un programa que permita ingresar y ordenar la siguiente numeración 33, 45, 67, 8, 12, 32, 56, 7, 9, 10, 22, 45, 98, 9. De forma ascendente y descendente, usando el método de Quicksort.

3.4. BÚSQUEDA SECUENCIAL

Una de la operación más común empleada con los arreglos es la búsqueda de elementos, y la técnica más elemental empleada para realizar este trabajo es la búsqueda secuencial. La búsqueda secuencial es una técnica en la cual se toma un valor clave (Elemento a buscar) y se empieza a comparar con todos los demás elementos, registro por registro, el resultado de la comparación es el índice o posición del elemento y en caso de no encontrar valor, el resultado puede definirse como nulo.

Del anterior término, se deriva su nombre de *Búsqueda secuencial,* pues se comparan secuencialmente todos los elementos, desde el inicio hasta el fin de arreglo uno por uno, hasta que el elemento del arreglo se encuentre o hasta que se llegue al final del arreglo. La existencia se puede asegurar desde el momento en que el elemento es localizado, pero no se puede asegurar la no existencia hasta no haber analizado todos los elementos del arreglo.

La búsqueda secuencial funciona de forma lineal y es muy útil en arreglos de pocos elementos, o bien para arreglos que no estar ordenados. Su

efectividad y velocidad se incrementa si los elementos de un arreglo ya se encuentran ordenados, esto es porque reduce el área de búsqueda eliminando los registros que están sobre el elemento clave.

Algoritmo:

```
Iniciar
I←1
Bandera←Falso
Repetir mientras (I<=N) y (Bandera=Falso)
        Si V [ I ] =X entonces Bandera=Verdadedo
                Si_no I=I+1
        Fin_si
Fin_Repetir
Si Bandera=Verdadero entonces
        Mostrar “Elemento en la Posición I”
Si_no
        Mostrar “Elemento no encontrado”
Fin_si
Fin_Inicio
```

ACTIVIDAD DE APRENDIZAJE

1. Realizar y entregar una investigación sobre el término búsqueda y búsquedas secuenciales.
2. Desarrolla un programa que permita ingresar siguiente 30 números. Luego debe permitir realizar la búsqueda de uno de los números, cualesquiera, empleando la búsqueda secuencial.

3.5. BÚSQUEDA BINARIA

Se utiliza cuando el vector en el que queremos determinar la existencia o no de un elemento está ordenado, o puede estarlo, este algoritmo reduce el tiempo de búsqueda considerablemente, ya que disminuye exponencialmente con el número de iteraciones.

Este algoritmo está altamente recomendado para buscar en arreglos enormes: En uno de 50 000 000 elementos, tarda 26 iteraciones en ejecutarse 1, suponiendo que la búsqueda falla, sino, siempre tarda menos en buscarlo.

Para implementar este algoritmo, se compara el elemento a buscar con un elemento cualquiera del arreglo (normalmente el elemento central), si el valor de éste es mayor que el del elemento buscado, se repite el procedimiento en la parte del arreglo que va desde el inicio de éste hasta el elemento tomado, en caso contrario se toma la parte del arreglo que va desde el elemento tomado hasta el final. De esta manera obtenemos intervalos cada vez más pequeños, hasta que se obtenga un intervalo indivisible, con el elemento buscado como elemento central. Si el elemento no se encuentra dentro de este último, entonces se deduce que el elemento buscado no se encuentra en el arreglo.

Si la tabla de números está ordenada, por ejemplo, en orden creciente, es posible utilizar para la búsqueda un algoritmo más eficiente que se basa en un concepto muy utilizado en la programación: dividir para vencer.

Si está ordenada la tabla y miramos el número situado en la mitad para ver si es mayor o menor que el número buscado (o con suerte igual), sabremos si la búsqueda ha de proceder en la subtabla con la mitad de tamaño que está antes o después de la mitad. Si se repite recursivamente el algoritmo al final o bien encontraremos el número sobre una tabla de un sólo elemento, o estaremos seguros de que no se encuentra allí.

```
Algoritmo: (N es el número de componente)
Iniciar
Izq←1
Der ←N
Bandera←Falso
Repetir mientras (Izq <= Der) y (Bandera=falso)
        Cen←Parte_Entera((Izq+Der)/2)
        Si X=V [Cen] entonces
                Bandera←Verdadero
        Si_no
                Si X>v[Cen] entonces
```

```
                Izq←Cen+1
            Si_no
                Der ← Cen-1
            Fin_si
        Fin_si
Fin_si
Si (Bandera=Verdadero) entonces
            Mostrar “Elemento en la posición Cen”
Si_No
            Mostrar ”Elemento no está en registro”
Fin_si
Fin_Inicio
```

ACTIVIDAD DE APRENDIZAJE

1. Realizar y entregar una investigación sobre el términos búsqueda binaria.
2. Desarrolla un programa que permita ingresar 30 números. Luego debe permitir realizar la búsqueda de uno de los números, cualesquiera, empleando la búsqueda binaria.

AUTOEVALUACIÓN

1. Es el proceso de localizar un registro con valor particular ( ) Shell

2. Es el proceso de arreglar los registros de tal manera que queden ordenados en base a un criterio o valor. ( ) QuickSort

3. Son los tipos de ordenamientos ( ) Método de Burbuja

4.- Este método consiste en acomodar los vectores moviendo el mayor hasta la última casilla comenzando desde la casilla cero, esto se logra comparando valores de llaves e intercambiarlos si no están en una posición relativa correcta ( ) Búsqueda secuencial

5.- Este método también se conoce con el nombre de inserción con incrementos decrecientes ( ) Arreglo y Archivos

6.- Este método se basa en la táctica "divide y vencerás" ( ) Búsqueda secuencial

7.- es la técnica más simple para buscar un elemento en un arreglo. Consiste en recorrer el arreglo elemento a elemento e ir comparando con el valor buscado (clave) ( ) Ordenamiento

8.- Se utiliza cuando el contenido del vector no se encuentra o no puede ser ordenado. ( ) Búsqueda

Respuesta: 5, 6, 4, 7, 3, 8, 2, 1