La enseñanza de los arrays estáticos, dinámicos y listas enlazadas ¿cuál usar? Análisis de códigos.

The teaching of static arrays, dynamics and linked lists. What to use? Code analysis.

Juan Carlos Fonden Calzadilla[1] Agustín Navarrete Herrera[2] Edgar Delfino Tchissingui[3]

[1]Universidad Tecnológica de La Habana, "José Antonio Echeverría",Cujae.

Correo electrónico:fonden1980@gmail.com.

[2]Universidad de Oriente

Correo electrónico: tinnavarrete@gmail.com

[3] Universidad Tecnológica de Bie "José Eduardo Dos Santos" en Angola

Correo electrónico:delfinotchissingui@gmail.com

Recibido: 18 de julio de 2018 Aceptado: 4 de diciembre de 2018

Resumen

El presente trabajo, encaminado a profesores de Programación, programadores, estudiantes de Ingeniería Informática y de Ciencias de la Computación, tiene como objetivo argumentar las potencialidades que brindan las listas enlazadas al compararlas con los arrays de tamaño estático y los arrays dinámicos, estructuras que históricamente han coexistido en los lenguajes de programación más empleados. Se obtuvieron, además, algoritmos para la solución de problemas, la modelación gráfica de las estructuras, códigos para la creación, impresión y búsqueda de elementos, con el uso de los lenguajes de programación Java y C# y un conjunto de recomendaciones de carácter metodológico, válidas para estudiantes, profesores de programación y programadores. Para ello se aplicaron métodos de investigación científica, entre ellos: El análisis documental, la modelación, enfoque de sistema, la observación, las tormentas de ideas unido a las experiencias de los autores como profesores de programación. Entre las conclusiones se expresa que, en el proceso de enseñanza y aprendizaje de los arrays, listas contiguas y listas enlazadas es recomendable escribir el algoritmo que soluciona el problema, luego modelar gráficamente la estructura y finalmente escribir el código en el lenguaje de programación y que las listas doblemente enlazadas, aunque consumen más espacio, ofrecen una mayor facilidad para recorrerlas al permitir el acceso secuencial en ambas direcciones.

Palabras clave: Programación, estructuras de datos, listas, listas enlazadas, arrays, arrays dinámicos

Abstract

The present work, directed to professors of Programming, programmers, students of Computer Engineering and of Sciences of the Computation, has the objective to argue the potentialities that offer the linked lists when comparing them with the arrays of static size and the dynamic arrays, structures that historically have coexisted in the most used programming languages. Algorithms were also obtained for the solution of problems, the graphic modeling of the structures, codes for the creation, printing, and search of elements, with the use of Java and C # programming languages and a set of recommendations of character methodological, valid for students, programming teachers and programmers. To this end, scientific research methods were applied, among them: documentary analysis, modeling, system approach, observation, storms of ideas together with the experiences of authors as programming teachers. Among the conclusions is expressed that, in the teaching and learning process of the arrays, contiguous lists and linked lists it is advisable to write the algorithm that solves the problem, then graphically model the structure and finally write the code in the programming language and Double-linked lists, although they consume more space, offer a greater facility for traversing them by allowing sequential access in both directions.

.

Key words: Programming, data structures, lists, linked lists, arrays, dynamic arrays

1. Introducción

En la actualidad, es frecuente que los estudiantes de ingenierías, al introducirse en el Proceso de Enseñanza y Aprendizaje de un lenguaje de programación, cometan errores al transformar su algoritmo a código fuente y al identificar las colecciones de datos más recomendadas para la solución de su problema. Estos errores pueden ser, a veces, irremediables, provocando un aumento considerable en su tiempo de aprendizaje, cansancio y posible deserción escolar al sentirse desalentados y considerar que la disciplina no es comprensible para ellos.
Por otro lado, la enseñanza de la programación en el contexto universitario, es para los profesores, un gran reto desde lo pedagógico, didáctico y metodológico.

La experiencia de los autores en diferentes universidades asevera que el proceso de enseñanza y aprendizaje de las Estructuras de Datos, en la cual se estudian los arrays, listas contiguas y enlazadas, es complejo y en general se obtienen bajos resultados académicos, cuestión que genera polémicas y preocupaciones a estudiantes, padres y directivos de las universidades. Por tales razones se continúa investigando, para ofrecer orientaciones metodológicas y nuevas ideas que enriquezcan y faciliten su Proceso de Enseñanza y Aprendizaje.

En la docencia, resolver un problema con el empleo arreglos (arrays) se justifica por la insuficiencia que tiene una variable simple de solo contener un dato. Si se desea almacenar varios datos sin que uno desplace al anterior, se necesitarían variables diferentes para lograrlo. Sin embargo, otras limitaciones en los arrays de tamaño estático provocaron la construcción de nuevos tipos de estructura de datos: Los arrays dinámicos o listas contiguas y las listas enlazadas. [1]
La solución eficiente de un problema docente en las disciplinas de Introducción a la Programación, Programación Orientada a Objetos y Estructuras de Datos, con el empleo de los arrays de tamaño estático, listas contiguas y enlazadas u otras estructuras de datos en el contexto universitario, ha sido, en ocasiones, una interrogante difícil de responder acertadamente. Las estructuras antes mencionadas tienen una relación costo - beneficio al crearla, insertar elementos, recorrerla, borrar, buscar un elemento y ordenarla. No existe la estructura perfecta para todas las operaciones, aunque si la más recomendada para uno u otro problema a resolver.

Al analizar los proyectos informáticos realizados por equipos de estudiantes en el Proceso de Enseñanza y Aprendizaje de la Programación Orientada a Objetos y Estructuras de Datos, junto a los softwares observados en la red de redes se pudo constatar el excesivo uso de las clases ArrayList en Java y List en C#. Ambas poseen como base un array de elementos, frente a la clase LinkedList, disponible en ambos lenguajes, que implementa una lista doblemente enlazada.

Por otro lado, aunque en las disciplinas Estructura de Datos I y II se estudian las listas enlazadas (nodos enlazados) simple, doble y circular; los árboles, colas de prioridad y grafos, estos contenidos resultan ser olvidados, poco o nunca usados en el desempeño profesional de estudiantes, especialistas y profesores de Programación, anteponiendo los arrays de tamaño estático y las listas contiguas.

Entonces, ¿Qué son realmente los arrays de tamaño estático y los arrays dinámicos o listas contiguas? ¿Qué insuficiencias tienen los arrays que dieron origen a las listas contiguas y enlazadas? ¿Cuál se debe usar para resolver un problema? ¿Qué criterios se deben tener en cuenta para el empleo eficiente de una u otra estructura de datos?

A continuación, se dará respuesta a las anteriores preguntas, con argumentos basados en la sistematización de los estudios publicados en diferentes revistas especializadas, monografías, libros electrónicos y otras fuentes bibliográficas, unido a la experiencia vivencial de los autores.

1.1 Arrays y arrays dinámicos

Un array es una estructura donde se guarda una colección finita de datos del mismo tipo. Se accede a cada elemento individual del array mediante un número entero denominado índice. Cero (0) es el índice del primer elemento y n-1 es el índice del último elemento, donde n, es la dimensión del array. [2] En el presente trabajo a este array se le llamará también array de tamaño estático, por que aceptará siempre una colección finita de datos.

A continuación, la representación gráfica de un array de tamaño estático donde se almacenan 10 números enteros, ubicados en posiciones desde la 0 hasta la 9.

1	**12**	**2**	**5**	**22**	**3**	**9**	**56**	**7**	**8**
0	1	2	3	4	5	6	7	8	9

Figura 1. Elaboración propia.

Características fundamentales de los arrays

Son colecciones de elementos homogéneos. Aceptan solo un tipo de datos en cada celda de una matriz o vector. Tanto en Java como en C# los arrays se declaran de forma análoga. En ambos lenguajes los arrays son objetos con características individuales y heredan de la clase base Object que es la raíz de todo el árbol de la jerarquía de clases del lenguaje y las construidas por los usuarios. En C# los arrays son objetos de la clase System.Array. La declaración es así: tipo [ ] identificador; lo que hace es declarar una referencia a un array. En Java existe la clase Arrays que implementa un conjunto de operaciones sobre los arrays. Se pueden crear arrays de objetos de cualquier tipo. La creación de un array, una vez que se conoce su identificador se hace de la siguiente forma: identificador = new tipo [cantidad de elementos]. Ejemplo int [ ] p = new int [5]. De esta forma se está reservando espacio para los elementos del array de tipo entero y el objeto p puede acceder a cada elemento del array, a través del índice.

La memoria asignada es contigua y estática esto puede ocasionar desperdicio de la misma. El programador no necesita estar al tanto de la siguiente o anterior asignación de memoria. Se puede acceder a los datos almacenados a través de un índice y se les puede recorrer con solo incrementarlo. Se les llaman estructuras indexadas. Proporcionan acceso aleatorio como array [1], array [4, 5]. La inserción y eliminación requieren más de tiempo que en una lista enlazada;

En Java y C#, la memoria de un array se asigna en tiempo de ejecución mediante el empleo del operador new. Java verifica el rango de un array y si detecta el ingreso o acceso a un elemento fuera de los índices establecidos, provoca que el programa rompa el tiempo de ejecución generando una excepción;

Cuando la memoria esté muy fraccionada, puede ser costoso o tal vez imposible encontrar espacio contiguo para ampliar un array dinámico. Los arrays consumen menos memoria al almacenar el mismo elemento que una lista enlazada porque ella adiciona, además del dato, al menos un campo dedicado a enlazar un nodo con otro. Se clasifican los arrays de tamaño estáticos según su dimensión en unidimensionales y multidimensionales [6]. La declaración de un array estático bidimensional se realiza de la forma:

tipo [ ][ ]nombre=new tipo[cantidad][cantidad].

Por otro lado, una lista enlazada es una estructura datos que representa una secuencia de valores, que pueden repetirse más de una vez. Si el mismo valor se repite varias veces, cada ocurrencia está considerada un elemento distinto. [2]

En las listas no enlazadas o arrays dinámicos (listas contiguas) los elementos se almacenan en posiciones consecutivas de memoria. Se corresponden con el concepto general de array de la mayoría de lenguajes de programación. A menudo se representa una lista como una sucesión de elementos entre paréntesis, separados por comas y pueden expresarse como L = (a0, a1, a2, ..., an−1), también se representa L = [a, b, c, d, e, ……x]. En este tipo de estructura de datos se identifican ventajas y desventajas en la programación de la solución de un problema. [3][4]

En las listas contiguas se insertan elementos sin un límite pre establecido, aunque su cambio de tamaño es una tarea costosa, implica copiar su contenido a una nueva zona más grande de memoria, y luego liberar el espacio del array anterior. La inserción o eliminación de un elemento, necesita una traslación de ellos, antes o después del insertado o eliminado.

Inicio Max

22	32	56	6	67	8						
0	1	2	3	4	5	6	7	8	9	10	11

Figura 2. Elaboración propia.

La figura 2 representa una lista contigua o array dinámico, en el que se adicionan elementos hasta llegar a un máximo permitido, luego para continuar adicionando se realiza una copia de ella. Este proceso se repite continuamente y puede ocupar memoria innecesaria. Así acontece en la clase ArrayList y List de Java y C# respectivamente.

1.2 Características de las listas enlazadas

Las listas enlazadas, son secuencias de nodos que se enlazan con apuntadores o referencias a direcciones de memoria “desarrolladas en 1955-56 por Cliff Shaw y Herbert Simon en RAND Corporation, como la principal estructura de datos para su Lenguaje de Procesamiento de la Información”, fue usado por los autores para desarrollar varios programas relacionados con la inteligencia artificial, incluida la Máquina de la Teoría General, el Solucionador de Problemas Generales, y un programa informático de ajedrez. [2]

En la actualidad, de acuerdo a los softwares observados en la red de redes, la tendencia a la creación de listas enlazadas por la gran mayoría de los programadores en Visual Basic, Java, Python y C# se realiza aprovechando las potencialidades de la Programación Orientada a Objetos, a través de la creación de clases y de referencias a direcciones de memorias de objetos de clases.

Las listas enlazadas poseen las siguientes características:

La información se almacena aleatoriamente, en nodos ubicados en diferentes partes de la memoria de acceso aleatoria (RAM). Basta con conocer la posición del primer nodo, que al menos necesita dos atributos, en uno se guarda la información (dato) y otro contiene la referencia a la dirección de memoria en el que se encuentra el siguiente nodo. Se accede a cada nodo secuencialmente y no de forma indexada, debido a la interdependencia o vínculo de ellos, por lo que para acceder al último nodo de la lista hay que recorrer los n-1 elementos previos. Se crea cada nodo cuando se hace una solicitud, esto es una ventaja sobre los arrays. La inserción y eliminación son simples y rápidas. Se pueden realizar por cualquier parte de la lista. Al inicio o al final, después o antes de un determinado nodo, además, se eliminan nodos dependiendo del campo de información o dato que se desea suprimir de la lista. Tienen la facilidad de expandirse y disminuir su tamaño, según se necesite agregar o eliminar elementos en tiempo de ejecución. La búsqueda en una lista enlazada no ordenada necesita que se recorran todos los nodos hasta encontrar el ítem que se busca antes de llegar a null. Si la lista enlazada está ordenada, siempre que el nodo actual sea mayor que el elemento que se busca, se puede detener la pesquisa automáticamente haciendo uso de un centinela o variable booleana inicializada en false, que cambiará a true cuando se cumpla la condición y se acelerará la pesquisa.

Representación gráfica de un nodo de una lista simplemente y doblemente enlazada:

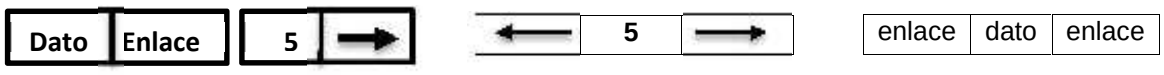

Figura 3. Elaboración propia.

A continuación, la figura 4 muestra una lista enlazada simple donde el último nodo apunta a null, lo cual significa el fin de la lista.

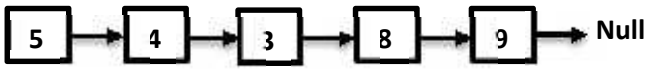

Figura 4. Elaboración propia.

Por otra parte, cuando los nodos tienen un doble enlace, es decir, un enlace al siguiente (dirección sucesora) y otro enlace al nodo anterior (dirección predecesora) como se muestra en la figura 5, entonces se conocen como lista doblemente enlazada.

Figura 5. Elaboración propia.

La lista doblemente enlazada aventaja a la lista simplemente enlazada en su recorrido en ambos sentidos, de izquierda a derecha como de derecha a izquierda, los nodos primero y último apuntan a null, que simboliza el final de la lista. El puntero anterior permitirá el acceso hacia el elemento anterior de la lista, mientras que el puntero siguiente permitirá el acceso hacia el próximo elemento. Su desventaja con relación a las listas simplemente enlazadas es que ocupan más memoria por nodo.

También existen las listas simples y doblemente enlazadas circulares donde el último nodo enlaza con el primero y este enlaza con el último. Así se representa:

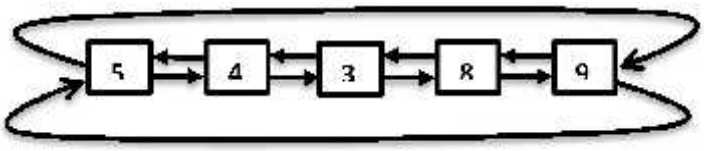

Figura 6. Elaboración propia.

En el presente trabajo se analizan las ventajas del empleo de las listas enlazadas sobre las listas contiguas o arrays dinámicos, en la solución de problemas docentes y además se elaboraron los algoritmos y códigos de programación, aplicables a los lenguajes de programación Java y C# para la realización de un conjunto de operaciones aplicables a las listas enlazadas. Se analizarán los códigos y algoritmos para la inserción de datos, buscar un elemento en la lista, y mostrar todo el contenido.

Como parte de la investigación realizada, en el Proceso de Enseñanza y Aprendizaje de los arrays y las listas enlazadas se han identificado varias insuficiencias que a continuación se presentan.

a) Para resolver los problemas relacionados con las antedichas estructuras de datos, los profesores no incluyen ejemplos tomados de la vida real, de las empresas u organizaciones, con el propósito de conquistar la atención del estudiante universitario;

b) Los estudiantes no han desarrollado previamente el pensamiento algorítmico, abstracto y relacional para planificar la solución de un problema, descomponer el todo en sus partes y luego recomponerlo y la realización de analogías para el aprendizaje;

c) Algunos profesores no estimulan a sus estudiantes al empleo de listas enlazadas para la solución de problemas.

2. Materiales y métodos

El trabajo investigativo que propició la elaboración de este artículo comenzó en 2012 en la facultad de Ingeniería Industrial de la Universidad Tecnológica de La Habana “José Antonio Echeverría” y continua en la Escuela Superior Politécnica de Bie. Angola, durante los cursos escolares 2016-2017, 2017-2018 y 2018-2019 en la carrera de Ingeniería Informática. Se consultaron diversas fuentes bibliográficas, entre ellas, sitios web dedicados a la programación, libros electrónicos y artículos relacionados con la didáctica de la enseñanza de la programación, enciclopedias, foros de programación en línea, libros impresos, monografías, softwares publicados en la red y tesis doctorales. La investigación fue posible, además, por la experiencia vivencial aportada por los autores como profesores e investigadores en las disciplinas Programación Orientada a Objetos y Estructuras de Datos en diferentes contextos universitarios. La Modelación como método se empleó al elaborar las secuencias de pasos o algoritmos y los códigos en el lenguaje de programación para insertar, imprimir y buscar. La Sistematización y el Análisis Documental de los estudios realizados sobre las colecciones más empleadas en la programación en los lenguajes de programación Java y C#, permitieron al autor identificar lo común y lo diferente entre los arrays y las listas enlazadas, así como la precisión de cuándo deben aplicarse cada una de ellas en la solución de un problema. La Observación a clases conferencias de Programación, algoritmos, más de 20 códigos de programación, clases prácticas y de laboratorios, permitió a los autores un mayor acercamiento a la creación de

códigos de arrays, arrays dinámicos con el empleo de las listas que ofrecen los lenguajes antes citados (ArrayList y List) y listas con el empleo de nodos enlazados.

Para el conocimiento de la evolución de las colecciones, entre ellas, las listas y los arrays, comenzando con los lenguajes de programación C y C++, y luego Java y C#, sus antecedentes, contradicciones y formas de programarlas en el presente, el autor se valió del método Histórico – Lógico. Los procedimientos lógicos del pensamiento Análisis y Síntesis, así como la Inducción y Deducción, proporcionaron a los autores elaborar conclusiones y recomendaciones relacionadas con el empleo de los arrays y las listas enlazadas, aplicables a estudiantes, profesores y programadores.

3. Resultados

Los principales resultados de las acciones desarrolladas en esta etapa de la investigación se resumen en la obtención de las modelaciones gráficas de los arrays de tamaño estático, dinámicos y las listas enlazadas; la creación de los algoritmos y códigos en el lenguaje Java para la creación de listas enlazadas, puntualizando en la inserción de datos, imprimir y buscar un elemento; los códigos y las recomendaciones metodológicas a estudiantes de Ingeniería Informática y Ciencias de la Computación, profesores de programación y programadores y la identificación de algunas dificultades que se producen en el Proceso de Enseñanza y Aprendizaje de los arrays y las listas enlazadas.

A continuación, se exponen los algoritmos y códigos para para la creación, búsqueda, borrado e impresión de los elementos de una lista doblemente enlazada en el lenguaje de programación Java, adaptable también al lenguaje de programación C#.

Se parte de una clase llamada Estudiante, que posee una clase interna llamada Nodo donde se encapsulan los atributos: ci, nombre, edad, siguiente y anterior. Se desea crear una lista enlazada que almacene los datos de un conjunto indeterminado de estudiantes. Se parte de la hipótesis que se realizarán continuadas inserciones y borrados de elementos en la lista.

3.1 Creación de algoritmos para el trabajo con listas enlazadas.

Algoritmo general para crear listas con nodos enlazados:

1. Crear: Una clase externa o clase contenedora llamad,,a Estudiante. Una clase anidada llamada Nodo dentro de la clase Estudiante. [7,8]

Designar, dentro de la clase interna Nodo las variables que recibirán los datos y las de tipo apuntadores o referencias (anterior e siguiente), de tipo Nodo. Inicializar los datos y las referencias en el constructor de la clase Nodo, las referencias se inicializan a null (siguiente=null y anterior=null). Crear dentro de la clase Estudiante y fuera de la clase interna Nodo las referencias primerNodo y ultimoNodo ambos de tipo Nodo. La referencia primerNodo contendrá la dirección del primer elemento de la lista y la referencia ultimoNodo contendrá la dirección del último elemento de la lista. Inicializar primerNodo y ultimoNodo dentro del constructor de la clase Estudiante (primerNodo=null y ultimoNodo=null). Crear los métodos de la clase Estudiante (Insertar, Mostrar, Buscar[9,10].

3.2 Creación de códigos en el lenguaje de programación Java

```
public class Estudiante {
```

```
 //Clase Nodo de Estudiante
 public class Nodo{
 //Atributos de la clase Nodo
 private int edad;
 private String nombre;
 private String ci;
 private Nodo siguiente, anterior;
```

```
 //Costructor de la clase Nodo
      public Nodo() {
       this.edad=0;
       this.nombre="";
       this. siguiente =null;
       this.anterior=null;
      }} //Fin de la clase Nodo
```

```
Nodo primerNodo, ultimoNodo;
      static int c=0;
//Construtor de la clase Estudiante
public Estudiante () {primerNodo=null; ultimoNodo=null;}
//Fin de la clase Estudiante
```

Algoritmo y código para insertar datos la derecha del primer nodo

Declarar un método con parámetros (definir tipos de parámetros).

Crear un objeto de tipo Nodo (nuevo) invocando su constructor sin parámetros. Con el objeto creado invocamos los atributos de la clase interna Nodo y le asignamos los datos pasados por parámetros. Si el primerNodo es igual a null o vacío, entonces primerNodo y ultimoNodo apuntan a nuevo nodo, sino: El apuntador siguiente del ultimoNodo apunta nuevo (el nuevo nodo). El apuntador anterior del nuevo Nodo apunta para el último. El nuevo nodo será el ultimoNodo de la lista, esperando otro nuevo.

```
public void insertarDatosDerecha(String ci, String nom, int ed){
        Nodo nuevo = new Nodo();
             nuevo.nombre=nom;
             nuevo.edad=id;
             nuevo.ci=ci;
             contadorNodos++;
       if (primerNodo==null) {
           primerNodo=ultimoNodo= nuevo;
       }
       else{
           ultimoNodo.siguiente = nuevo;
           nuevo.anterior=ultimoNodo;
           ultimoNodo= nuevo;
       }}
```

Se crea un nodo que recibe los datos entrado por parámetros (ci), (nom) y (ed). A ese nodo se le llama nuevo. Si primerNodo apunta a null entonces se hace que primero apunte a nuevo y si no es así, entonces se comienza a insertar a la derecha haciendo que ultimoNodo.siguiente apunte a nuevo, que nuevo.anterior apunte a ultimoNodo y siempre el último nodo apunta a nuevo.

Figura 7. Inserción de nodos a la derecha del primero. Elaboración propia.

Algoritmo y código para insertar datos a la izquierda del primer nodo

1. Declarar un método con parámetros (definir tipos de parámetros). Crear un objeto de tipo la clase interna Nodo (nuevo) invocando su constructor sin parámetros. Con el objeto creado invocamos los atributos de la clase interna Nodo

y le asignamos los datos pasados por parámetros. Si el primerNodo apunta a null o vacío entonces: El primerNodo y ultimoNodo apuntan para nuevo (el nuevo nodo) sino es así: El apuntador anterior del primerNodo apunta para nuevo (el nuevo nodo). El apuntador siguiente de nuevo apunta para el primerNodo. El nuevo nodo será el primerNodo de la lista

```
//Método para insertar elementos en la lista
public void insertarDatosIzquierda(String ci,String nom, int id){
        Nodo nuevo = new Nodo();
             nuevo.nombre=nom;
             nuevo.edad=id;
             nuevo.ci=ci;
             c++;
       if (primerNodo==null) {
           primerNodo=ultimoNodo= nuevo;
       }
       else{
           primerNodo.anterior= nuevo;
           nuevo. siguiente =primerNodo;
           primerNodo= nuevo;
       }}
```

Se crea un nodo que recibe los datos entrados por parámetros carnet de identidad, nombre y edad. A ese nodo se le llama nuevo. Si primerNodo apunta a null entonces se hace que primerNodo apunte a nuevo y si no es así, entonces se comienza a insertar a la izquierda haciendo que primerNodo.anterior apunte a nuevo, que nuevo. Siguiente apunte a primer Nodo y finalmente el primerNodo apunta a nuevo.

Figura 8. Inserción de nodos a la izquierda del primero. Elaboración propia.

En el artículo trabajo no se ha trabajado con los métodos de accesos de cada atributo para racionalizar espacio en el mismo.

Algoritmos y códigos para imprimir datos del primero al último

Declarar un método con retorno de tipo String sin parámetros. Crear un objeto de tipo Nodo llamado percorre y se inicializa en primerNodo o en ultimoNodo y una variable do tipo String llamada mostrar inicializada en " ".

Crear una estructura repetitiva (while) que iterará mientras la variable de percurso (percorre) sea diferente de null. Concatenar (mostrar+=percorre. datos, de acuerdo a los datos que se deseen mostrar). La variable percurso recibe percurso.siguiente o percurso.anterior para las próximas iteraciones de ciclo

//Método para mostrar la lista

while. Retornar la variable mostrar.

```
    public String mostrarDatos(){
        Nodo percorre=primerNodo; String mostrar="";
        while(percorre!=null){
 mostrar+=percorre.getCi()+percorre.getNombre()+ percorre.getEdad();
         percorre=percorre.siguiente;
        } return mostrar;}
```

Algoritmos y códigos para buscar datos en la lista no ordenada

Algoritmo:

Crear: Un método con parámetro (dato a buscar) con retorno (boolean). Una variable unEstudiante de tipo Nodo (inicializada en el primerNodo). Un ciclo (while) que iterará mientras la variable de unEstudiante sea diferente de null. Una estructura condicional (if).

Si el dato a buscar es igual a un dato encontrado en la lista entonces retornar true. Cerrar el ciclo. Si no lo encontró retornar false

//Método para buscar un elemento en la lista

```
public boolean BuscarSegunCI (String CI)
{
     Nodo unEstudiante=primerNodo;
       while(unEstudiante!=null) {
       if( unEstudiante.ci.equalsIgnoreCase(CI))
     return true;
        unEstudiante=unEstudiante.siguiente;
        }return false;}
```

3.3. Recomendaciones a estudiantes, profesores y programadores

La elaboración de un buen algoritmo, es una tarea tan atrayente y necesaria como la creación de un programa con el empleo de un lenguaje determinado.

Por tales razones, para profesores, estudiantes o programadores es aconsejable reconocer las siguientes interrogantes antes de iniciar un algoritmo: ¿Qué datos o elementos iniciales se poseen? ¿Qué se desea obtener? ¿Qué otros datos

necesitarían? ¿Cómo alcanzar el resultado? ¿Qué fórmula(s) debo emplear? ¿Qué otro procesamiento se necesita? ¿Qué pasos dar?

Analiza el problema que pretendes resolver, identifica las posibles vías de solución y verifica si necesitarás de un array con tamaño estático, array dinámico o una lista enlazada para resolverlo.

El profesor debe dialogar más sobre los conceptos de arrays estáticos, dinámicos y las listas contiguas o enlazadas, reflexionar sobre los mismos, no cansarse de repetir las ideas de diversos modos, comparar las diferentes estructuras, establecer relaciones entre ellas, hacer preguntas reflexivas, derivar conclusiones parciales, modelar gráficamente las soluciones, elaborar y explicar algoritmos y finalmente concluir con el código de programación.

Para resolver los problemas relacionados con las antedichas estructuras de datos, los profesores deben vincularlas con situaciones tomadas de las empresas u organizaciones, con el propósito de estimular la atención del estudiante universitario.

Si en el problema que has de resolver identificas un número finito de elementos a almacenar y recorrer, emplear una lista enlazada será una mala práctica de programación. Empleas un array de tamaño estático.

Tanto para recorrer un array como una lista contigua, puedes emplear la estructura repetitiva for (tipo variable: array) en Java o en C# el foreach (tipo variable in arrray). Ambas estructuras repetitivas fueron diseñadas para recorrer colecciones. En C#, las clases List o ArrayList tienen una funcionalidad similar, pero se considera que List ofrece un mayor rendimiento y tiene seguridad de tipos.

En Java, si se identifica que habrá muchas búsquedas en listas grandes, es aconsejable implementar la interface List instanciándola en un ArrayList. Ejemplo: List listaElementos = new ArrayList().

Si por lo contrario, las operaciones que más se realizarán serán las inserciones y eliminaciones en listas grandes, entonces es aconsejable implementar la interface List instanciándola en un LinkedList. Ejemplo: List listaElementos = new LinkedList().

4. Discusión

El estudio realizado, desde la fase exploratoria demostró la necesidad de profundizar en la conceptualización, y determinación de los elementos comunes y divergentes entre los arrays de tamaño estáticos, [11,12] dinámicos y las listas enlazadas.

La opinión de profesores y estudiantes de segundo año de la carrera de Ingeniería Informática durante tres cursos escolares en la provincia de Bie – Angola, es que el trabajo con listas enlazadas contribuyó al desarrollo de su pensamiento lógico, algorítmico y abstracto ya que para resolver un problema no se emplean las bondades que suministran las bibliotecas de clases de los lenguajes de programación, sino que, las posibles soluciones son primeramente modeladas gráficamente, luego algoritmizadas y finalmente resueltas en el lenguaje de programación, y la creación de las listas enlazadas, precisa de un pensamiento imaginativo, capaz de representar simbólicamente una estructura y la solución de un problema.

En el presente trabajo se presentaron algunos algoritmos y códigos para crear, recorrer y buscar un nodo en una lista doblemente enlazada, partiendo de un ejemplo hipotético de una clase llamada Estudiante en donde habrá continuas inserciones y borrados de elementos.

Se emplearon clases anidadas, pueden existir varios niveles de anidamiento. Una clase interna es otro miembro más de la clase contenedora o clase externa. Las clases anidadas permiten la creación de un código más legible y compacto, y aumentan la encapsulación ya que pueden ocultarse de otras distintas a la clase que la contiene, aunque pueden acceder a atributos de su clase exterior sin que estos dejen de ser privados.

Por otra parte, para garantizar un doble recorrido de las listas, [13] hacia adelante y/o hacia atrás, o dado un elemento, conocer rápidamente el anterior y siguiente, entonces, lo más recomendado fue trabajar con listas doblemente enlazadas.

Los arrays dinámicos en Java y C#, ofrecen acceso en tiempo constante a los elementos de la lista, pero si deseas añadir o borrar en cualquier posición que no sea la última es necesario mover los elementos. Además, si el arreglo ya está lleno es necesario crear uno nuevo con mayor capacidad y copiar los elementos existentes para continuar la introducción de datos. Una lista contigua es más

apropiada para cantidades pequeñas de datos, donde la memoria continua esté siempre disponible.

Las listas enlazadas aventajan a las listas contiguas en que la inserción y extracción de nodos se realizan con un gasto de recursos independiente al tamaño de la lista. No hay necesidad de grandes cantidades de memoria contigua. El uso de memoria se adapta dinámicamente al número de datos almacenados en la lista en cada momento. [13, 14, 15].

5. Conclusiones

En el proceso de enseñanza y aprendizaje de los arrays, listas contiguas y listas enlazadas es recomendable escribir el algoritmo que soluciona el problema, luego modelar gráficamente la estructura y finalmente escribir el código en el lenguaje de programación [16].

Diseñar algoritmos, aunque se clasifiquen como simples es casi siempre difícil para la mayoría de los estudiantes. Según algunos profesores esto se debe a diversos factores, tales como el poco empleo estrategias de autoaprendizaje, el poco desarrollo del pensamiento lógico y la insuficiente vinculación del contenido con problemas reales de las empresas u organizaciones [17].

Los arrays se emplean cuando se conoce de antemano la cantidad de elementos que se almacenarán, y esa cantidad no corre el riesgo de ser alterada en tiempo de ejecución del programa. Tanto las listas contiguas como las enlazadas se emplean cuando se desconoce la cantidad de elementos con las que trabajará un programa. La selección del tipo de lista (simple, doble, circular) depende de las exigencias del problema, el lenguaje de programación y los conocimientos del programador.

Las listas enlazadas convienen ser empleadas antes que las listas contiguas dinámicas y los arrays con tamaño estático cuando en una aplicación se necesita realizar cuantiosas operaciones de inserción y eliminación de elementos. Las listas doblemente enlazadas, aunque requieren más espacio por nodo por causa de sus referencias siguiente y anterior, ofrecen una mayor facilidad para recorrerlas al permitir el acceso secuencial en ambas direcciones.

La enseñanza de las Estructuras de Datos, exige de una minuciosa preparación metodológica, por parte de los educadores que imparten esta disciplina, para alcanzar mejores resultados docentes y evitar deserciones en los estudiantes.

Referencias bibliográficas

1. Archundia E, Cerón C, Boone M. Estructuras de Datos en Computación: Aprendizaje mediado a través del Diseño Digital Centrado en el Usuario. Revista Iberoamericana de Producción Académica y Gestión Educativa. 2015; vol (no): pp
2. Oracle. Arrays. [Internet]. Java Documentation. The Java Tutorial. 2017 [citado junio de 2017]. Disponible en: https://docs.oracle.com
3. Microsoft. Clase genérica List. [Internet] 2018 [citado octubre 2018]. Disponible en: https://msdn.microsoft.com
4. Ferreira A, Rojo G. Enseñanza de la programación. Revista Iberoamericana de Tecnología en Educación y Educación en Tecnología [Internet]. 2001 [citado noviembre 2018]; 1(1): pp. Disponible en: http://teyet-revista.info.unlp.edu.ar/TEyET/article

5. Ayala J, Aguilar I, Hidalgo A, Gómez H. Memoria dinámica en el lenguaje de programación C [Internet]. España: Universidad de la Rioja; 2016 [citado octubre 2018]. Disponible en: https://dialnet.unirioja.es/descarga/libro/652144.pdf
6. Vaca C. Estructuras de Datos y Algoritmos [Internet]. España: Universidad de Valladolid; 2011 [citado octubre 2018]. Disponible en: https://www.infor.uva.es
7. Peinado F. Las clases anidadas [Internet]. Madrid: Departamento de Ingeniería del Software e Inteligencia Artificial. Facultad de Informática. Universidad Complutense de Madrid; 2018 [citado diciembre 2018]. Disponible en: https://www.fdi.ucm.es/profesor
8. Microsoft. Clase genérica List. [Internet] 2018 [citado octubre 2018]. Disponible en: https://msdn.microsoft.com
9. Oracle. When to Use Nested Classes, Local Classes, Anonymous Classes, and Lambda Expressions [Internet]. Java Documentation. The Java Tutorial. 2018 [citado octubre 2018]. Disponible en: https://docs.oracle.com
10. Román J. Listas enlazadas. Programación de Sistemas [Internet]. Madrid: Universidad Carlos III de Madrid; 2018 [citado noviembre 2018]. Disponible en: www.cartagena99.com/recursos

11. Didáctica y divulgación de la programación. Listas doblemente encadenadas Java [Internet]. Foros aprenderaprogramar.com; 2015 [citado noviembre 2018]. Disponible en: https://aprenderaprogramar.com
12. Bruno RO. Un enfoque práctico y didáctico para el diseño de algoritmos. Programación y Algoritmia [Internet] 2017 [citado octubre 2018]. Disponible en: www.utn.edu.ar/static/
13. Difference between Array and Linked List. [Internet] Techdifferences; 2018 [citado noviembre 2018]. Disponible en: https://techdifferences.com/difference-between-array-and-linked-list.html

14. Dalal A, Atri A . An introduction to Linked List. International Journal of Research (IJR) [Internet] 2014 [citado noviembre 2014]; 1(10): pp. Disponible en: https://edupediapublications.org/journals/

15. Ayala J, Aguilar I, Alfonso J, Hidalgo Z and Gómez H. Memoria dinámica en el lenguaje de programación C [Internet]. España: Editorial CENID; 2016 Recuperado de https://dialnet.unirioja.es ISBN: 978-607-8435-19-7.

16. Díaz E. J. "Herramienta para facilitar el aprendizaje de listas enlazadas en c++ utilizando realidad aumentada en dispositivos móviles" Trabajo de grado. Línea de investigación: Inteligencia computacional. (2017). Recuperado de http://repositorio.unicartagena.edu.co:8080

17. Insuasti J. Problemas de enseñanza y aprendizaje de los fundamentos de programación. Revista educación y desarrollo social, 10 (2), 234-246. DOI: org/10/18359/reds.1701. (2016) Recuperado de https://dialnet.unirioja.es

Autores
Juan Carlos Fonden Calzadilla Licenciado en Educación en la Especialidad de Matemática, Doctor en Ciencias Pedagógicas, Profesor Auxiliar Universidad Tecnológica de La Habana "José Antonio Echeverría", Cujae.
Agustín Navarrete Herrera. Master en Nuevas Tecnologías para la Educación, Licenciado en Ciencias de la Computación. Asistente Profesor Universidad de Oriente.
Edgar Delfino Tchissingui Estudiante. Alumno ayudante de 4to año de la Carrera de Ingeniería Informática Universidad Tecnológica de Bie "José Eduardo Dos Santos" en Angola.