LISTAS ENLAZADAS

Maria C. Torres Madroñero
Profesora asociada
Departamento de Ciencias de la Computación y la Información

Objetivos

- Implementar las clases para el nodo simple y lista simple
- Implementar las clases para el nodo doble y lista doble
- Implementar una solución algorítmica empleando colecciones de objetos, haciendo uso de la estructura de datos lista simple y lista doble

Recursos requeridos

- PC
- IDE para JAVA o Python – el estudiante deberá seleccionar un lenguaje de programación para el desarrollo de las prácticas de laboratorio

Actividades preliminares al laboratorio

- Lectura de la guía

Marco teórico

ESTRUCTURA DE DATOS CON MEMORÍA DINÁMICA

Los arreglos no son la única forma para manipular colecciones de datos durante el tiempo de ejecución de un programa. Existes otras estructuras de datos, que a diferencia de los arreglos, permiten el uso de memoria dinámica. Recordemos que los arreglos tienen la ventaja de permitir la indexación directa a los diferentes elementos que los componen; sin embargo, esta indexación requiere la definición de un a tamaño fijo de la colección de datos al principio del tiempo de ejecución. Las estructuras de datos que emplean memoria dinámica permiten administrar colecciones de datos que van cambiando de tamaño durante el tiempo de ejecución, de acuerdo con la demanda del recurso. Entre las estructuras de datos con memoria dinámica encontramos las listas (listas simples, listas dobles), pilas, colas y árboles.

NODO SIMPLE

Para introducir la estructura de datos lista simple, partiremos de la definición de una estructura denomina Nodo Simple. Un nodo simple es una abstracción que permite almacenar un dato y un enlace hacia otro nodo simple. El dato que se almacena en el nodo ser un dato de tipo primitivo (int, floar, double) o un objeto (ej. Usuario, fecha, hora). El enlace hacia otro nodo almacena un apuntador hacia la posición de memoria donde el siguiente nodo se encuentra, es decir, este apuntador crea una conexión unidireccional entre dos nodos. La Figura 1 presenta una representación gráfica del concepto de nodo simple.

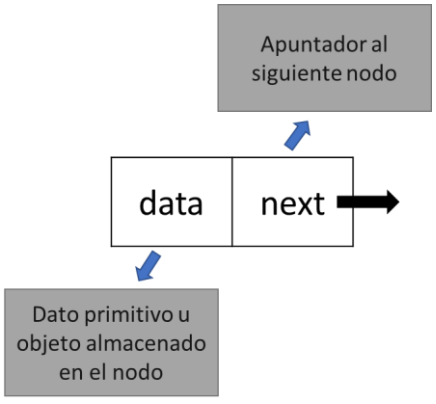


Figura 1: Nodo simple - representación gráfica.

La implementación de un nodo simple requiere de una clase con al menos dos atributos: uno dedicado al almacenamiento de los datos y otros que mantiene el apuntador al siguiente nodo. El diagrama de clase de la Figura 2 presenta la clase Node (nodo simple).

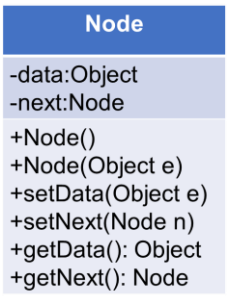


Figura 2. Clase Node - Nodo simple

LISTA SIMPLE

Una lista simple o lista enlazada simple es una estructura de datos que emplea memoria dinámica. Al ser una estructura de datos permite la administración de una colección de datos durante el tiempo de ejecución del programa. A diferencia de los arreglos, las listas simples cambian de tamaño en el tiempo de ejecución.

Una lista simple es una colección de nodos simples enlazados a través del atributo next (siguiente nodo). Las listas simples no permiten un acceso directo a cualquier posición de la colección. Este tipo de estructuras tiene un acceso secuencial, es decir tenemos acceso al primer dato directamente, y a partir de ese nodo nos podemos mover por la estructura empleando el atributo next. La principal ventaja de esta estructura de datos es el uso de la memoria, ya que no se requiere definir un tamaño para la lista al inicio del tiempo de ejecución.

La Figura 3 presenta la representación gráfica para una lista enlazada simple.

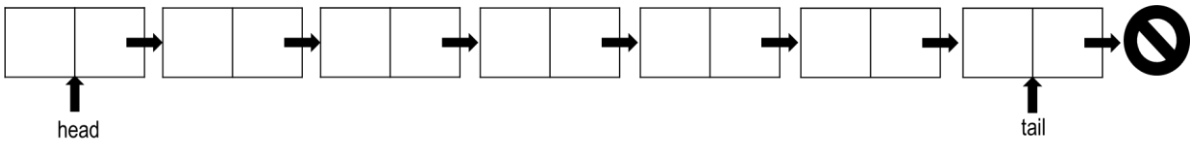


Figura 3. Lista simple - representación grafica.

La implementación de la clase List (lista simple) se presenta en el diagrama de clase de la Figura 4.

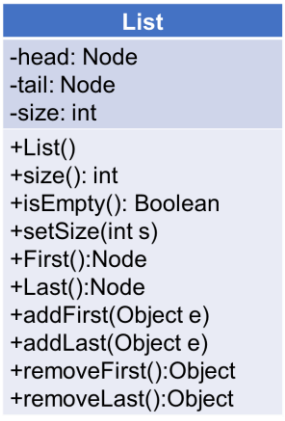


Figura 4. Diagrama de clase List (Lista simple)

Para la clase lista simple se incluyen dos atributos:

- head: es un nodo simple (objeto tipo Node) que apunta al primer nodo de la lista simple o cabecera. A través de este atributo se accede a toda la colección de datos.
- tail: es un nodo simple (objeto tipo Node) que apunta al último nodo de la lista simple o cola, este atributo es opcional y permite simplificar algunas operaciones.
- size: es un atributo de tipo entero que indica el número de datos en la colección – es decir el número de nodos

Tanto los atributos head y tail se pueden inicializar en nulo, y el tamaño o size en cero.

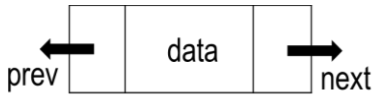
Los métodos sugeridos que debe incluir la clase lista simple son:

- size(): retorna el número de elementos en la lista simple – número de datos en la colección
- isEmpty(): retonar verdadero (TRUE) si la lista está vacía, es decir que la colección no tiene datos; retorna FALSO si el tamaño es diferente de cero.
- setSize(): método opcional que permite modificar de forma externa el tamaño para operaciones de agregar nuevos nodos o datos a la colección
- First(): retorna la cabecera de la lista simple
- Last(): retorna el último nodo de la lista simple o cola
- addFirst(): inserta un objeto (dato de tipo primitivo u objeto) al principio de la lista simple
- addLast(): inserta un elemento al final de la lista simple
- removeFirst(): elimina y retorna el ultimo elemento de la lista

NODO DOBLE

A diferencia del nodo simple, el nodo doble incluye un enlace adicional que permite conectarse al nodo previo. Esto permite simplificar las operaciones para ingresar o eliminar nodos en puntos intermedios de una lista. La representación grafica del nodo doble se presenta en la figura, así como su respectivo diagrama de clase. En este caso tenemos tres atributos:

- Data: dato u objeto almacenado en el nodo
- Next: apuntador al siguiente nodo doble
- Prev: apuntador al nodo doble previo



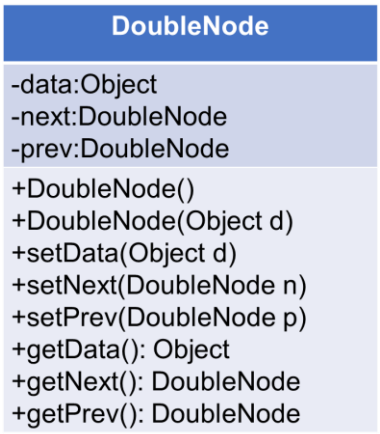


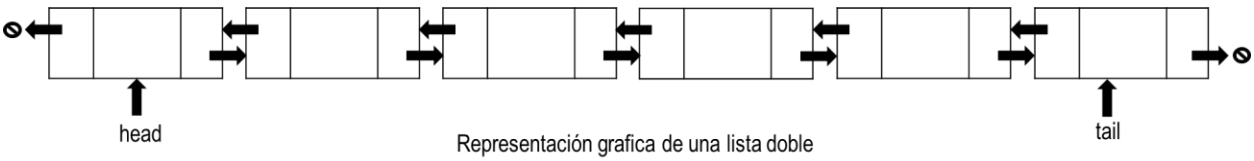
Figura 5. Diagrama de clase DoubleNode

LISTA DOBLE

La clase lista doble es una conexión sucesiva de nodos dobles. A diferencia de los arreglos, las listas usan memoria dinámica, esto es usan o liberan memoria en la medida que se crean o eliminan datos. Y a diferencia de la lista simple, la lista doble permite desplasarnos desde la cabeza a la cola, o al revés. Pero también, por las conexiones dobles,

facilita las operaciones de insertar y eliminar nodos intermedios de la lista. Para la clase definimos los siguientes atributos:

- La cabecera (head) que permite acceder al primer nodo de la lista
- La cola (tail) que permite acceder al último nodo de la lista (atributo opcional)
- El tamaño (size) de la lista que mantiene el número de nodos en la colección



Representación grafica de una lista doble

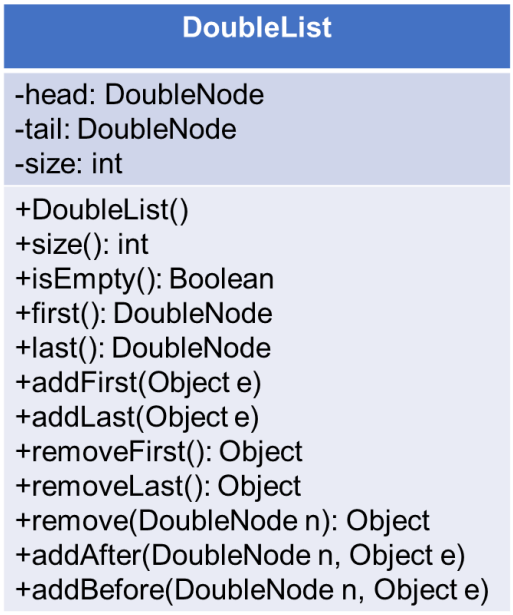


Figura 5. Diagrama de clase DoubleList

Los métodos que se incluyen para la clase DoubleList son:

- DoubleList(): constructor vacio, inicializa la cabezara y cola como Null y el tamaño de la lista en 0
- Size(): retorna el tamaño de la lista, es decir, el número de nodos
- isEmpty(): retorna verdadero si la lista esta vacía, falso si tiene algún elemento
- first(): retorna un apuntador a la cabecera de la lista
- last(): retorna un apuntador a la cola de la lista
- addFirst(Object e): crea un nuevo nodo donde se almacena el objeto que ingresa como paramero, el nuevo nodo es insertado al principio de la lista
- addLast(Object e): crea un nuevo nodo donde se almacena el objeto que ingresa como paramero, el nuevo nodo es insertado al final de la lista
- removeFirst(): elimina el nodo que se encuentra al principio de la lista, retornando el elemento almacenado
- removeLast(): elimina el nodo que se encuentra al final de la lista, retornando el elemento almacenado
- remove(DoubleNode n): elimina el nodo n de la lista doble, retornando el elemento almacenado
- addAfter(DoubleNode n, Object e): crea un nuevo nodo con el dato que ingresa como parámetro, inserta el nuevo nodo después del nodo n
- addBefore(DoubleNode n, Object e): crea un nuevo nodo con el dato que ingresa como parámetro, inserta el nuevo nodo antes del nodo n.

EJEMPLO DE TABLERO DE PUNTAJES EMPLEANDO LISTA SIMPLE

En el laboratorio 5 estudiamos el tablero de puntajes basado en arreglos. Este problema también lo podemos solucionar empleando una lista simple. La Figura 5 presenta el diagrama de clase para esta solución.

ScoreBoardList()
- board: List
+ScoreBoardList( ) +add(GameEntry e) +remove(String n): GameEntry +toFile()

Las principales diferencias de esta implementación respecto a la basada en arreglos, anteriormente estudiada, incluye:

- En el caso de la lista simple no necesitamos un atributo de tipo entero que nos índice cuantos puntajes validos tenemos almacenado. Esto se debe a que la clase lista simple (List) incluye el atributo size, que nos indica cuantos datos tenemos.
- El constructor de SocreBoardList() no recibe ningún parámetro de entrada. En el caso de la implementación con arreglos, recordemos que ingresábamos la capacidad o tamaño del arreglo como parámetro del constructor. En el caso de listas, como estas cambian de tamaño en el tiempo de ejecución del programa, no requerimos este parámetro.
- El método remove() de la implementación con listas simples la realizamos basada en el nombre del jugador, y no de una posición como en la implementación con arreglos. Esto se debe a que en las listas simples no podemos indexar posiciones específicas.

A continuación, se describen cada uno de los seudocodigos:

```
//Seudocodigo para el constructor
ScoreBoard( ) //constructor vacio
     board = new List()



//Seudocodigo para agregar un nuevo puntaje
add(GameEntry e)
   Node n = new Node(e)
   if board.isEmpty()
      board.addFirst(e)
   else
      Node temp = board.First()
      while (temp!=null && temp.getData().getScore()>e.getScore())
            Node anterior = temp
            temp = temp.getNext()
      if temp==null
            board.addLast(e)
      else
            anterior.setNext(n)
            n.setNext(temp)
            board.setSize(board.size()++)

//Seudocodigo para eliminar un puntaje
remove(String n)
   Node temp = board.First()
```

```
while(temp!=null&&temp.getData().getName()!=n)
      Node anterior = temp
      temp = temp.getNext()
if temp!=null
      anterior.setNext(temp.getNext)
      temp.setNext(null)
      board.setSize(board.getSize()--)
```

ACTIVIDADES

1. Implementar las clases nodo simple y lista simple
2. Implementar las clases nodo doble y lista doble
3. Validar el funcionamiento de su listas simple y doble con los siguientes problemas (se deben implementar para cada lista):
    a. Crear una colección donde cada nodo almacena los números pares del 1 al 20, el programa debe imprimir los datos en pantalla. Posteriormente, se debe eliminar los números 1, 10 y 20; nuevamente el programa debe mostrar los datos en la lista resultante.
    b. Crear una colección con al menos 5 usuarios (use la clase usuario de los laboratorios anteriores). El programa debe imprimir los datos en pantalla. Posteriormente, debe pedir por consola un nuevo usuario para ser insertado al principio, otro para ser insertado al final de la lista. Nuevamente el programa debe mostrar los usuarios en la lista. Solo para la lista doble, el programa debe pedir un usuario e insertarlo después del tercer nodo, para ello debe usar el método addBefore o addAfter.

Instrucciones de entrega

- La solución de los problemas debe desarrollarse en JAVA o Python. Los estudiantes tendrán la libertad de seleccionar el lenguaje de programación y plataforma para presentar la solución de los problemas.
- La solución debe emplear librerías nativas y se invita a los estudiantes a no usar código descargado de internet. Los laboratorios están diseñados para practicar los fundamentos teóricos; entre más código escriba el estudiante más fácil será su comprensión de los temas de clase.
- La solución se puede presentar en grupos de hasta 3 estudiantes.
- La solución de los problemas debe entregarse y sustentarse en el aula de clase o en la hora de asesoría a estudiantes. No se reciben soluciones por correo electrónico.