Listas doblemente enlazadas

8.9. LISTA DOBLEMENTE ENLAZADA

Hasta ahora, el recorrido de una lista se ha realizado en sentido directo *(adelante)*. Existen numerosas aplicaciones en las que es conveniente poder acceder a los elementos o nodos de una lista en cualquier orden, tanto hacia adelante como hacia atrás. En este caso, se recomienda el uso de una **lista doblemente enlazada.** En esta lista, cada elemento contiene dos punteros (referencias), además del valor almacenado. Una referencia apunta al siguiente elemento de la lista y la otra

referencia apunta al elemento anterior. La Figura 8.11 muestra una lista doblemente enlazada y un nodo de dicha lista.

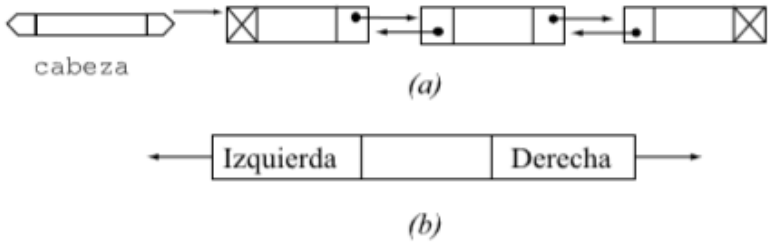

Figura 8.11 Lista doblemente enlazada. (a) Lista con tres nodos; (b) nodo

Las operaciones de una *Lista Doble* son similares a las de una *Lista: insertar, eliminar, buscar, recorrer...* La operación de insertar un nuevo nodo en la lista debe realizar ajustes de los dos *pointer*. La Figura 8.12 muestra el problema de insertar un nodo a la derecha del nodo *actual;* como se observa, se asignan cuatro enlaces.

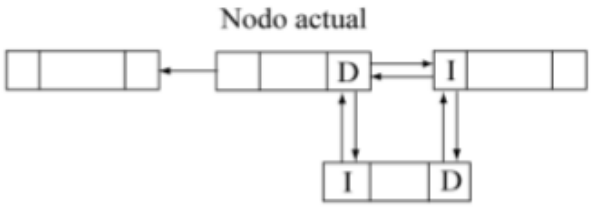

Figura 8.12 Inserción de un nodo en una lista doblemente enlazada

La operación de eliminar (borrar) un nodo de la lista doble necesita enlazar, mutuamente, el nodo anterior y el nodo siguiente del que se borra, como se observa en la Figura 8.13.

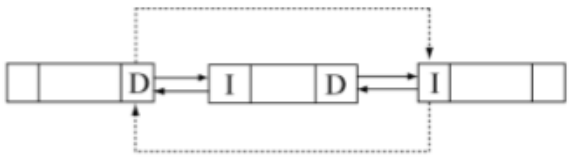

Figura 8.13 Eliminación de un nodo en una lista doblemente enlazada

8.9.1. Nodo de una lista doblemente enlazada

Un nodo de una lista doblemente enlazada tiene dos punteros (referencias) para enlazar con los nodos izquierdo y derecho, además de la parte correspondiente al campo dato. La clase `Nodo` agrupa los componentes del nodo de una lista doble; por ejemplo, para datos de tipo entero:

```
package listaDobleEnlace;

public class Nodo
{
   int dato;


     Nodo adelante;
     Nodo atras;
        // ...
  }
```

El constructor asigna un valor al campo `dato` y las referencias `adelante, atras` se inicializan a `null`.

```
public Nodo(int entrada)
{
   dato = entrada;
   adelante = atras = null;
}
```

8.9.2. Insertar un elemento en una lista doblemente enlazada

La clase `ListaDoble` encapsula las operaciones básicas de las listas doblemente enlazadas. La clase dispone de la variable `cabeza` que referencia el primer nodo de la lista, permite acceder a cualquier otro nodo. El constructor de la clase inicializa la lista vacía `(null)`.

Se puede añadir un nuevo nodo a la lista de distintas formas, según la posición donde se inserte. Naturalmente, el algoritmo empleado para añadir varía dependiendo de la posición en que se desea insertar el elemento. La posición de inserción puede ser:

- En la *cabeza* (elemento primero) de la lista.
- Al final de la lista (elemento último).
- Antes de un elemento especificado.
- Después de un elemento especificado.

Insertar un nuevo elemento en la cabeza de una lista doble

El proceso sigue estos pasos:

1. Crear un nodo con el nuevo elemento y asignar su referencia a la variable `nuevo`.
2. Hacer que el campo enlace `adelante` del nuevo nodo apunte a la cabeza (primer nodo) de la lista original, y que el campo enlace `atras` del nodo cabeza apunte al nuevo nodo.
3. Hacer que `cabeza` apunte al nuevo nodo que se ha creado.

A continuación, se escribe el método, miembro de la clase `ListaDoble`, que implementa la operación.

Código Java

```
public ListaDoble insertarCabezaLista(Elemento entrada)
{
   Nodo nuevo;
   nuevo = new Nodo(entrada);
   nuevo.adelante = cabeza;
   if (cabeza != null )
     cabeza.atras = nuevo;
   cabeza = nuevo;
   return this;
}
```

Insertar después de un nodo

La inserción de un nuevo nodo se puede realizar también en un nodo intermedio de la lista. El algoritmo de la operación que inserta después de un nodo `n`, requiere las siguientes etapas:

1. Crear un nodo con el nuevo elemento y asignar su referencia a la variable `nuevo`.
2. Hacer que el enlace `adelante` del nuevo nodo apunte al nodo siguiente de `n` (o bien a `null` si `n` es el último nodo). El enlace `atras` del nodo siguiente a `n` (si `n` no es el último nodo) tiene que apuntar a `nuevo`.
3. Hacer que el enlace `adelante` del nodo `n` apunte al nuevo nodo. A su vez, el enlace `atras` del nuevo nodo debe de apuntar a `n`.

El método `insertaDespues()` implementa el algoritmo, supone que la lista es de números enteros, y naturalmente es un método de la clase `ListaDoble`. El primer argumento, `anterior`, representa el nodo `n` a partir del cual se enlaza. El segundo argumento, `entrada`, es el dato que se añade a la lista.

Código Java

```
public ListaDoble insertaDespues(Nodo anterior, Elemento entrada)
{
   Nodo nuevo;

   nuevo = new Nodo(entrada);
   nuevo.adelante = anterior.adelante;
   if (anterior.adelante != null)
      anterior.adelante.atras = nuevo;
   anterior.adelante = nuevo;
   nuevo.atras = anterior;
   return this;
}
```

8.9.3. Eliminar un elemento de una lista doblemente enlazada

Quitar un nodo de una lista doble supone realizar el enlace de dos nodos, el nodo *anterior* y el nodo *siguiente* al que se desea eliminar. La referencia `adelante` del nodo anterior debe apuntar al nodo *siguiente,* y la referencia `atras` del nodo *siguiente* debe apuntar al nodo *anterior.* La memoria que ocupa el nodo se libera automáticamente en el momento que éste deja de ser referenciado *(garbage collection*, recolección de basura).

El algoritmo es similar al del borrado para una lista simple. Ahora, la dirección del nodo *anterior* se encuentra en la referencia `atras` del nodo a borrar. Los pasos a seguir son:

1. Búsqueda del nodo que contiene el dato.
2. La referencia `adelante` del nodo anterior tiene que apuntar a la referencia `adelante` del nodo a eliminar (si no es el nodo *cabecera*).
3. La referencia `atras` del nodo siguiente a borrar tiene que apuntar a la referencia `atras` del nodo a eliminar (si no es el último nodo).
4. Si el nodo que se elimina es el primero, `cabeza`, se modifica `cabeza` para que tenga la dirección del nodo `siguiente`.
5. La memoria ocupada por el nodo es liberada automáticamente.

El método que implementa el algoritmo es miembro de la clase `ListaDoble`:

```
public void eliminar (Elemento entrada)
{
   Nodo actual;
   boolean encontrado = false;
   actual = cabeza;
   // Bucle de búsqueda
   while ((actual != null) && (!encontrado))
   {
        /* la comparación se realiza con el método equals()...,
         depende del tipo Elemento */
      encontrado = (actual.dato == entrada);
      if (!encontrado)
         actual = actual.adelante;
   }
    // Enlace de nodo anterior con el siguiente
   if (actual != null)
   {
      //distingue entre nodo cabecera o resto de la lista
      if (actual == cabeza)
      {
         cabeza = actual.adelante;
         if (actual.adelante != null)
            actual.adelante.atras = null;
      }
      else if (actual.adelante != null)    // No es el último nodo
      {
         actual.atras.adelante = actual.adelante;
         actual.adelante.atras = actual.atras;
      }
      else   // último nodo
         actual.atras.adelante = null;
      actual = null;
   }
}
```

Ejercicio 8.3

Crear una lista doblemente enlazada con números enteros, del 1 al 999, generados aleatoriamente. Una vez creada la lista, se eliminan los nodos que estén fuera de un rango de valores leídos desde el teclado.

En el apartado 8.9 se han declarado las clases `Nodo` y `ListaDoble`, formando parte del paquete `listaDobleEnlace`, necesarias para este ejercicio. Además, a la clase `Nodo` se añade el método `getDato()` para devolver el dato del nodo.

Para resolver el problema que plantea el ejercicio, en el paquete `listaDobleEnlace` se define la clase `IteradorLista`, para acceder a los datos de cualquier lista doble (de enteros). En el constructor de la clase `IteradorLista` se asocia la lista a recorrer con el objeto iterador. La clase `IteradorLista` implementa el método `siguiente()`; cada llamada a `siguiente()` devuelve el nodo actual de la lista y avanza al siguiente. Una vez que se termina de recorrer la lista, devuelve `null`.

La clase con el método `main()` crea un objeto que genera números aleatorios, los cuales se insertan en la lista doble. A continuación, se pide el rango de elementos a eliminar; con el objeto iterador se obtienen los elementos, y aquellos fuera de rango se borran de la lista (llamada al método `eliminar`).

```
// archivo con la clase Nodo de visibilidad public
package listaDobleEnlace;

public class Nodo
{
   // declaración de nodo de lista doble
   public int getDato()
   {
      return dato;
   }
}

// archivo con la clase ListaDoble de visibilidad public

package listaDobleEnlace;

public class ListaDoble
{
   Nodo cabeza;
   // métodos de la clase (implementación en apartado 8.9)
   public ListaDoble(){;}
   public ListaDoble insertarCabezaLista(int entrada){;}
   public ListaDoble insertaDespues(Nodo anterior, int entrada){;}
   public void eliminar (int entrada){;}
   public void visualizar() {;}
   public void buscarLista(int destino) {;}

}

// archivo con la clase IteradorLista de visibilidad public

package listaDobleEnlace;

public class IteradorLista
```

```
public class IteradorLista
{
   private Nodo actual;
   public IteradorLista(ListaDoble ld)
   {
      actual = ld.cabeza;
   }
   public Nodo siguiente()
   {
      Nodo a;
      a = actual;
      if (actual != null)
      {
         actual = actual.adelante;
      }
      return a;
   }
}
```

```
/*
   Clase con método main(). Crea el objeto lista doble e inserta
   datos enteros generados aleatoriamente.
   Crea objeto iterador de lista, para recorrer sus elementos y
   aquellos fuera de rango se eliminan. El rango se lee del teclado.
*/

import java.util.Random;
import java.io.*;
import listaDobleEnlace.*;

class ListaEnRango
{
   public static void main(String [] ar) throws IOException
   {
      Random r;
      int d, x1, x2;
      final int M = 29;  // número de elementos de la lista
      final int MX = 999;
      BufferedReader entrada = new BufferedReader(
                               new InputStreamReader(System.in));
      ListaDoble listaDb;
      r = new Random();
      listaDb = new ListaDoble();
      for (int j = 1; j <= M ; j++)
      {
         d = r.nextInt(MX) + 1;
         listaDb.insertarCabezaLista(d);
      }

      System.out.println("Elementos de la lista original");
      listaDb.visualizar();
      // rango de valores
      System.out.println("\nRango que va a contener la lista");
      x1 = Integer.parseInt(entrada.readLine());
      x2 = Integer.parseInt(entrada.readLine());
      // crea iterador asociado a la lista
      IteradorLista iterador = new IteradorLista(listaDb);
      Nodo a;
```

```
        Nodo a;
        // recorre la lista con el iterador
        a = iterador.siguiente();
        while (a != null)
        {
           int w;
           w = a.getDato();
           if (!(w >= x1 && w <= x2))  // fuera de rango
              listaDb.eliminar(w);
           a = iterador.siguiente();
        }

        System.out.println("Elementos actuales de la lista");
        listaDb.visualizar();
     }
  }
```

Referencias bibliográficas

Libro: Estructuras de datos en Java

Autores: Luis Joyanes Aguilar Ignacio Zahonero Martínez

Edificio Valrealty, 1. ª Planta

Basauri, 17 28023 Aravaca (Madrid)

Editor: José Luis García Técnico

Editorial: Blanca Pecharromán

Compuesto en: Gesbiblo, S. L.

Diseño de cubierta: Gesbiblo, S. L.