

Estructuras de Datos Clase 7 – Listas e Iteradores (segunda parte)

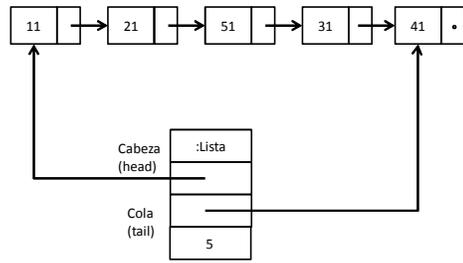


Dr. Sergio A. Gómez
http://cs.uns.edu.ar/~sag



Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Bahía Blanca, Argentina

Lista simplemente enlazada con enlace al principio y al final



La referencia al último elemento permite computar last() y addLast(e) en tiempo constante. También usaremos un campo para mantener la longitud y calcular size() en O(1).

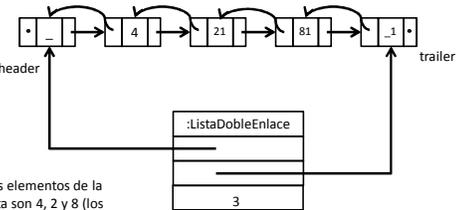
Listas con referencia al primer y último nodo

- Lista simplemente enlazada
- Se mantiene una referencia al primer y último elemento
- **Ventaja:**
 - addLast(e) y last() tienen orden 1.
- **Desventajas:**
 - Hay casos especiales cuando se elimina al principio y al final, sobre todo cuando la lista mide 1
 - prev(p) sigue siendo de orden lineal en la cantidad de elementos de la lista (hay que recorrer desde el comienzo)

Estructuras de datos - Dr. Sergio A. Gómez 3

Lista doblemente enlazada

- Cada nodo conoce un elemento, el nodo anterior y el nodo siguiente.
- La lista tiene dos nodos ficticios (dummy) llamados *celda de encabezamiento*, *header* y *trailer*, que evitan casos especiales al insertar y eliminar.
- La posición es directa, se conoce el primer y último nodo dummy.



Los elementos de la lista son 4, 2 y 8 (los nodos dummy son transparentes al cliente)

Estructuras de datos - Dr. Sergio A. Gómez 4

Listas doblemente enlazadas

- Lista doblemente enlazada con referencia al primer y último nodo y dos celdas de encabezamiento (una al principio y otra al final).
- **Ventajas:**
 - Cada nodo conoce el siguiente nodo y al nodo anterior
 - Todas las operaciones tienen orden 1
 - Al usar celdas de encabezamiento las operaciones no tienen casos especiales (e.g., casos con referencias nulas).
- **Desventajas:**
 - Mayor uso de espacio
- **Leer:** secciones 3.3 y 6.2.4 de Goodrich & Tamassia

Estructuras de datos - Dr. Sergio A. Gómez 5

Ejemplo de operación

Las operaciones no tienen casos especiales (la clase DNode denota a los nodos):

```
public void addAfter(Position<E> p, E element)
    throws InvalidPositionException {
    DNode<E> pos = checkPosition(p);
    DNode<E> nuevo = new DNode<E>(element);
    nuevo.setNext(pos.getNext());
    nuevo.setPrev(pos);
    nuevo.getNext().setPrev(nuevo);
    pos.setNext(nuevo);
    size++;
}
```

Nota: T_{addAfter}(n) = O(1)
Nota: Ver que funciona bien incluso cuando p==first y cuando p == last().

Estructuras de datos - Dr. Sergio A. Gómez 6

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente: "Estructuras de Datos. Notas de Clase". Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.

Iteradores

- Un iterador es un patrón de diseño que abstrae el recorrido de los elementos de una colección
- Un iterador consiste de una secuencia S , un elemento corriente S y una manera de avanzar al siguiente elemento de S haciéndolo el nuevo elemento corriente
- ADT Iterador (provisto por la interfaz `java.util.Iterator`)
 - `hasNext()`: Testea si hay elementos para recorrer en el iterador
 - `next()`: Retorna el siguiente elemento del iterador
- ADT Iterable: Para poder ser iterable una colección debe brindar el método:
 - `iterator()`: Retorna un iterador para los elementos de la colección

Estructuras de datos - Dr. Sergio A. Gómez

7

Lista Iterable

Para tener una lista que soporte la iteración debemos tener dos métodos adicionales:

```
public interface PositionList<E> extends java.util.Iterable<E> {
    // Todos los otros métodos que ya estudiamos
    ....
    // Más dos nuevos:

    // Retorna un iterador para los elementos de la lista:
    // Esta operación es requerida por java.util.Iterable
    public Iterator<E> iterator();

    // Devuelve una colección iterable de posiciones:
    public Iterable<Position<E>> positions();
}
```

Estructuras de datos - Dr. Sergio A. Gómez

8

Bucle for-each de Java

La sentencia for-each permite expresar un recorrido de una colección en alto nivel. Supongamos que "colección" es un tipo que implementa la interfaz `Iterable<E>`:

```
for( E elem : colección )
    sentencia(elem);
```

Permite ejecutar sentencia sobre cada elemento `elem` de colección. Además, el for-each equivale a:

```
Iterator<E> it = colección.iterator();
while( it.hasNext() )
{
    E elem = it.next();
    sentencia(elem);
}
```

Estructuras de datos - Dr. Sergio A. Gómez

9

Ejemplo: Hallar el máximo elemento de una lista de enteros positivos

```
static int hallarMaximo( PositionList<Integer> lista )
{
    int maximo = 0;
    for( Integer elem : lista )
        if( elem > maximo )
            maximo = elem;
    return maximo;
}

T_hallarMaximo(n) = O(n)
```

Estructuras de datos - Dr. Sergio A. Gómez

10

Ejemplo: Buscar un elemento x en una lista l

Solución no estructurada:

```
public static <E> boolean buscar(PositionList<E> l, E x) {
    for( E e : l )
        if( e.equals(x) ) return true;
    return false;
}
```

Solución estructurada:

```
public static <E> boolean buscar(PositionList<E> l, E x) {
    Iterator<E> it = l.iterator();
    boolean encontré = false;
    while( it.hasNext() && !encontré )
        if( it.next().equals(x) )
            encontré = true;
    return encontré;
}
```

Estructuras de datos - Dr. Sergio A. Gómez

11

Ejemplo: Método toString para lista

Usa el iterador de `PositionList` para devolver un string con el formato $[x_1, x_2, \dots, x_n]$ a partir de la lista receptora en $O(n)$.

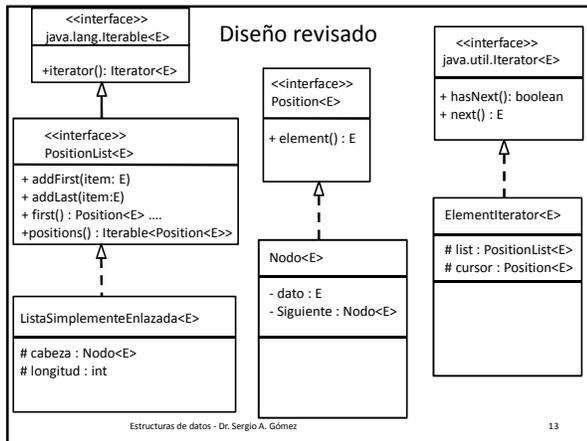
```
import java.util.Iterator;

public String toString() {
    Iterator<E> it = iterator(); // Le pido el iterador a la lista this.
    String s = "[";
    while( it.hasNext() ) {
        s += it.next(); // cast implícito de E a String, equivale a: s+=it.next().toString();
        if( it.hasNext() ) // Append de una coma si quedan elementos
            s += ",";
    }
    s += "]";
    return s;
}
```

Estructuras de datos - Dr. Sergio A. Gómez

12

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente: "Estructuras de Datos. Notas de Clase". Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.



Implementación:
Método iterator de la clase ListaSimplementeEnlazada

Delegaremos en la clase ElementIterator:

```

import java.util.Iterator;

public Iterator<E> iterator(){
    // Creo un elementIterator sobre la lista this a iterar
    return new ElementIterator<E>( this );
}
    
```

Estructuras de datos - Dr. Sergio A. Gómez 14

Clase ElementIterator

```

import java.lang.*;
import java.util.*;
public class ElementIterator<E> implements Iterator <E> {
    protected PositionList<E> list; // Lista a iterar
    protected Position<E> cursor; // Posición del elemento corriente

    public ElementIterator (PositionList <E> l) {
        list = l; // Guardo la referencia a la lista a iterar
        if (list.isEmpty()) cursor = null; // Si la lista está vacía, la posición corriente es nula
        else cursor = list.first(); // Si no la posición corriente es la primera de la lista
    }
    public boolean hasNext() { return cursor != null; } // Hay siguiente si el cursor no está más allá de la última posición
    public E next () throws NoSuchElementException {
        if ( cursor == null ) // Si el cursor es null, el cliente no testeó que hasNext fuera true
            throw new NoSuchElementException ("Error: No hay siguiente");
        E toReturn = cursor.element(); // Salvo el elemento corriente
        cursor = (cursor == list.last()) ? null : list.next(cursor); // Avanzo a la siguiente posición
        return toReturn; // Retorno el elemento salvado
    }
    public void remove() { /* No lo haremos. De hacerlo: remueve el último ítem retornado por next(), no se puede llamar a remove() hasta que no se haya ejecutado otro next(). Hay que agregar más control en las otras operaciones. */ }
}
    
```

Estructuras de datos - Dr. Sergio A. Gómez 15

Patrón adaptador (Adapter)

- **Objetivo:** Implementar una pila usando una lista.
- El patrón de diseño Adaptador permite usar una clase para brindar la funcionalidad de otra clase.
- Forma de usarlo:
 - implementar una pila mediante un atributo de tipo array list / PositionList
 - Cada operación de pila se implementa con una operación de array list (es decir, la pila *delega* en array list/PositionList).

Estructuras de datos - Dr. Sergio A. Gómez 16

Adaptador de pilas (Implementación de pila con PositionList)

Método de la pila	Implementación con PositionList I
size()	l.size()
isEmpty()	l.isEmpty()
push(x)	l.addFirst(x)
pop()	l.remove(l.first())
top()	l.first().element()

Tarea:

- Implementar una cola usando una PositionList.

Estructuras de datos - Dr. Sergio A. Gómez 17

Listas de Java: interfaz List

- Java brinda una versión de listas por medio de la interface java.util.List en las que sus posiciones se referencian por enteros 0 a n-1
- Implementa una secuencia $L=[x_0, x_1, x_2, \dots, x_{n-1}]$ de elementos de tipo genérico E
- n es la longitud de L
- La posición de cada elemento es un entero i entre 0 y n-1 tal que la posición de x_0 es 0, la de x_1 es 1, ..., la de x_{n-1} es n-1.
- Hay dos implementaciones ArrayList y LinkedList.

Estructuras de datos - Dr. Sergio A. Gómez 18

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente: "Estructuras de Datos. Notas de Clase". Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.

Listas de Java: ADT ArrayList

- `size()`: Retorna la cantidad de elementos de la lista S
- `isEmpty()`: Retorna verdadero si la lista S está vacía y falso en caso contrario
- `get(i)`: Retorna el elemento i-esimo de la lista S; ocurre un error si $i < 0$ o $i > \text{size}() - 1$
- `set(i,e)`: Reemplaza con e al elemento i-esimo; ocurre un error si $i < 0$ o $i > \text{size}() - 1$
- `add(i,e)`: Agrega un elemento e en posición i; ocurre un error si $i < 0$ o $i > \text{size}()$
- `remove(i)`: Elimina el elemento i-esimo de la lista S; ocurre un error si $i < 0$ o $i > \text{size}() - 1$

Estructuras de datos - Dr. Sergio A. Gómez

19

Ejemplo de ArrayList

Operación	Salida	S
<code>add(0, 7)</code>	-	[7]
<code>add(0, 4)</code>	-	[4, 7]
<code>get(1)</code>	7	[4, 7]
<code>add(2, 2)</code>	-	[4, 7, 2]
<code>get(3)</code>	error	[4, 7, 2]
<code>remove(1)</code>	7	[4, 2]
<code>add(1, 5)</code>	-	[4, 5, 2]
<code>add(1, 3)</code>	-	[4, 3, 5, 2]
<code>add(4, 9)</code>	-	[4, 3, 5, 2, 9]
<code>get(2)</code>	5	[4, 3, 5, 2, 9]
<code>set(3, 8)</code>	2	[4, 3, 5, 8, 9]

Estructuras de datos - Dr. Sergio A. Gómez

20

Adaptador

Método de la pila	Implementación con ArrayList I
<code>size()</code>	<code>l.size()</code>
<code>isEmpty()</code>	<code>l.isEmpty()</code>
<code>push(x)</code>	<code>l.add(l.size(), x)</code>
<code>pop()</code>	<code>l.remove(l.size()-1)</code>
<code>top()</code>	<code>l.get(l.size()-1)</code>

Tarea:

- Implementar una cola usando un ArrayList.
- Leer Secciones 6.1.3 a 6.1.5 (páginas 224-228) de Goodrich & Tamassia para ver cómo implementar un ArrayList usando un arreglo.

Estructuras de datos - Dr. Sergio A. Gómez

21

Clase java.util.ArrayList

- `size()`: Retorna la cantidad de elementos de la lista S
- `isEmpty()`: Retorna verdadero si la lista S está vacía y falso en caso contrario
- `get(i)`: Retorna el elemento i-esimo de la lista S; ocurre un error si $i < 0$ o $i > \text{size}() - 1$
- `set(i,e)`: Reemplaza con e al elemento i-esimo; ocurre un error si $i < 0$ o $i > \text{size}() - 1$
- `add(i,e)`: Agrega un elemento e en posición i; ; ocurre un error si $i < 0$ o $i > \text{size}()$
- `remove(i)`: elimina el elemento i-esimo de la lista S; ; ocurre un error si $i < 0$ o $i > \text{size}() - 1$
- `clear()`: Elimina todos los elementos de la lista
- `toArray()`: retorna un array con los elementos de la lista en el mismo orden
- `indexOf(e)`: índice de la primera aparición de e en la lista
- `lastIndexOf(e)`: índice de la última aparición de e en la lista

Estructuras de datos - Dr. Sergio A. Gómez

22

Bibliografía

- Goodrich & Tamassia, capítulo 6.

Estructuras de datos - Dr. Sergio A. Gómez

23

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente: "Estructuras de Datos. Notas de Clase". Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.