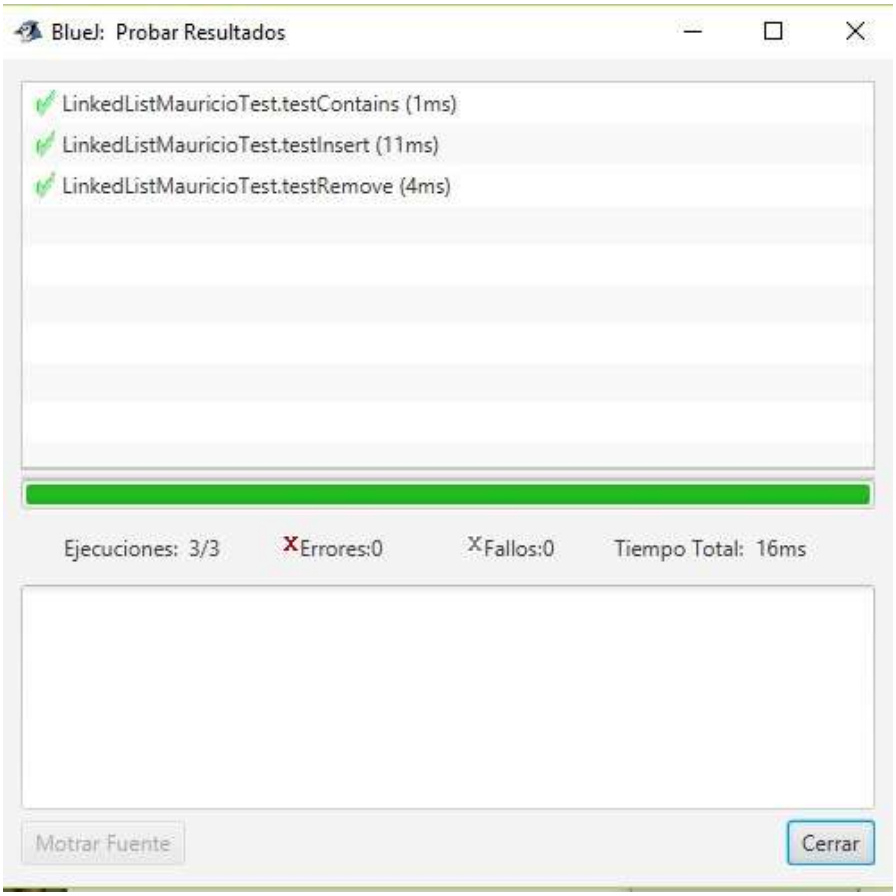
Laboratorio Nro. 4: Implementación de Listas Enlazadas

Luis Carlos Rodríguez Zúñiga
Universidad Eafit
Medellín, Colombia
lcrodriguz@eafit.edu.co

Laura Sánchez Córdoba
Universidad Eafit
Medellín, Colombia
lsanchezc@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

1. 

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

2. El algoritmo funciona de la siguiente manera: Para empezar se crea un ArrayList que tendrá datos de tipo Stack (Pilas) y un tamaño n que será definido por el usuario según entre en los datos, a partir de allí el Scanner leerá la entrada del usuario donde primero leerá un int que será el tamaño de nuestro objeto y posteriormente pasa a leer cada una de las instrucciones que digite el usuario hasta que lea “quit”, lo cual indica el fin de los comandos para que devuelva el resultado. Mientras no lea el comando que lo finaliza, éste leerá las entradas de la siguiente manera: un primer String (inst1) que indica la primera parte del comando, luego un int a indicando la posición donde el brazo empezará a trabajar, seguida por otro String (inst2) y otro int b que indica la otra posición con la que el brazo ejecutará el comando.

Una vez obtenidos estos datos se pasa a evaluar las instrucciones dependiendo y de cuales sean éstas se ejecutará un método:

1. Si la primera instrucción es “move”:

   1.1. Si la segunda instrucción es “onto” se llama al método moveOnto, el cual recibe como parámetros a a y b (los números de los bloques que el brazo moverá) y se ejecuta así: primero verifica que a y b son diferentes, de lo contrario dejará el orden de los bloques igual. Recorre todo el Arraylist para buscar si estos valores están en alguna de las pilas del Arraylist y los almacena en dos variables booleanas, por las que posteriormente se verifica que ambos elementos no estén en la misma pila, luego hallándose en la posición donde está b se crea una variable int pX que tomará el valor del elemento de encima de la pila y mientras no sea b lo regresará a su posición de origen.

   Si se halla donde está a, y se encuentran más bloques además del correspondiente al número a, se realiza el mismo proceso con la variable pX para posteriormente colocar el bloque a sobre b.

   1.2. Si la segunda instrucción es “over” se llama al método moveOver, el cual recibe como parámetros a a y b (los números de los bloques que el brazo moverá) y se ejecuta así: primero verifica que a y b son diferentes, de lo contrario dejará el orden de los bloques igual. Recorre todo el Arraylist para buscar si estos valores están en alguna de las pilas del Arraylist y los almacena en dos variables booleanas, por las que

posteriormente se verifica que ambos elementos no estén en la misma pila, luego hallándose en la posición donde está a se crea una variable int pX que tomará el valor del elemento de encima de la pila y mientras no sea a lo regresará a su posición de origen. Finalmente toma el bloque a y lo pone sobre la pila en la que se halla b.

2. Si la primera instrucción es “pile”:

    2.1. Si la segunda instrucción es “onto”:
    Se llama al método pileOnto, el cual recibe como parámetros a a y b (los números de los bloques que el brazo moverá) y se ejecuta así: primero crea dos variables enteras para guardar la posición de a y b (pA y pB respectivamente), verifica que a y b son diferentes, de lo contrario dejará el orden de los bloques igual. Recorre todo el Arraylist para buscar si estos valores están en alguna de las pilas del Arraylist y los almacena en dos variables booleanas, luego hallándose en la posición donde está a, se le asigna a pA el valor del índice (en el cual encontró que estaba ubicado el bloque a) y es el mismo proceso para b; una vez que se verifique que no están en la misma pila pasa lo siguiente:

    Se crea una variable int pX que tomará el valor del elemento de encima de la pila donde está b y mientras no sea b mismo, lo regresará a su posición de origen. Luego crea una pila auxiliar para guardar los elementos que estaban apilados sobre a y finalmente se almacena a. De ahí se pasa a un ciclo que colocará los elementos de la pila auxiliar sobre b.

    2.2. Si la segunda instrucción es “over” se llama al método pileOver, el cual recibe como parámetros a a y b (los números de los bloques que el brazo moverá) y se ejecuta así: primero crea dos variables enteras para guardar la posición de a y b (pA y pB respectivamente), verifica que a y b son diferentes, de lo contrario dejará el orden de los bloques igual. Recorre todo el Arraylist para buscar si estos valores están en alguna de las pilas del Arraylist y los almacena en dos variables booleanas, luego hallándose en la posición donde está a, se le asigna a pA el valor del índice (en el cual encontró que estaba ubicado el

bloque a) y es el mismo proceso para b; una vez que se verifique que no están en la misma pila pasa lo siguiente:

Se crea una pila auxiliar para guardar los elementos que estaban apilados sobre a y una variable int pX que tomará el valor del elemento de encima de la pila donde está a para almacenarlos uno a uno en la pila auxiliar y finalmente se almacena a. De ahí se pasa a un ciclo que colocará todos los elementos de la pila auxiliar sobre b.

*El método imprimir simplemente crea una nueva pila que agrega uno a uno los elementos del ArrayList con sus respectivas pilas e imprime una por una qué quedo en ellas.

3.

```
import java.io.*;
import java.util.*;
public class Punto2
{
   //basado en:
   //Ejercicio21
   //Autores: Daniel Arango y Kevin Parra
   //Fecha: Octubre 2017
   //Disponible en: https://github.com/damesaa201710054010/ST0245-033.git

   private ArrayList<Stack> laPila = new ArrayList<>();
   private int tam;

   public Punto2(int n) {
      tam = n; //c1
      for (int i = 0; i < tam; i++) {//c2*n
         Stack pila = new Stack(); //c3
         pila.push(i); //c4
         laPila.add(pila); //O(n)
      }
   }
T(n)=c+n*c2*(n)
O(n)=c+n*c2*(n)
O(n)=n*c2*(n) R.S.
```

O(n)=n*(n) R.P.
O(n)=n^2 R.P.

```
    public static void main(String[] args) throws IOException {
        Scanner in = new Scanner(System.in); //c1
        int n = in.nextInt(); //c2
        Punto2 ej = new Punto2(n); // O(n^2)
        String inst1 = in.next();
        while (!inst1.equals("quit")) {
            int a = in.nextInt(); //c3
            String inst2 = in.next(); //c4
            int b = in.nextInt(); //c5

            if (inst1.equals("move")) { //c5
                if (inst2.equals("onto")) { //c6
                    ej.moveOnto(a, b); //O(m*n)
                } else {
                    ej.moveOver(a, b); //O(m*n)
                }
            } else if (inst1.equals("pile")) { //c7
                if (inst2.equals("onto")) { //c8
                    ej.moveOnto(a, b); //O(m*n)
                } else {
                    ej.pileOver(a, b); //O(m*n)
                }
            }
            inst1 = in.next(); //c9
        }
        System.out.println(); //c10
        ej.imprimir(); //O(m*n)
    }
```

T(n) = c1+ n^2+((m*n)*5)
O(n) = c1+ n^2+((m*n)*5)
O(n) = n^2+((m*n)*5) R.S.
O(n) = (m*n)*5 R.S
O(n) = m*n R.P.

```
    private void moveOnto(int a, int b) {
```

```
        if (a != b) { //c1
            for (int i = 0; i < tam; i++) { //n*c2
                boolean loctA = laPila.get(i).contains(a);
                boolean loctB = laPila.get(i).contains(b);
                if (loctA != loctB) { //c3
                    if (loctB) { //c4
                        int pX = (int) laPila.get(i).pop(); //O(1)
                        while (pX != b) { //m*c5
                            laPila.get(pX).push(pX); //O(1)
                            pX = (int) laPila.get(i).pop(); //O(1)
                        }
                        laPila.get(i).push(b); //O(1)
                        laPila.get(i).push(a); //O(1)

                    }
                    if (loctA) { //c6
                        int pX = (int) laPila.get(i).pop(); //O(1)
                        while (pX != a) { //m*c7
                            laPila.get(pX).push(pX);//O(1)
                            pX = (int) laPila.get(i).pop(); //O(1)
                        }
                    }
                }
            }
        }
    }
T(n)= c+ n*c2*(m*c5+m*c7)
O(n)= c+ n*c2*(m*c5+m*c7)
O(n)= n*c2*(m*c5+m*c7) R.S.
O(n)= n*c2*(m*(c5+c7))
O(n)= n*c2*(m) R.P.
O(n)= n*m  R.P.


    private void moveOver(int a, int b){
        if (a!=b){ //c1
            for (int i = 0; i < tam; i++) { //n*c2
                boolean loctA = laPila.get(i).contains(a); //O(1)
                boolean loctB = laPila.get(i).contains(b); //O(1)
                if (loctB!=loctA){ //c3
                    if (loctA){ //c4
```

```
                int pA=(int)laPila.get(i).pop(); //O(1)
                while(pA!=a){ //m*c5
                    laPila.get(pA).push(pA); //O(1)
                    pA=(int)laPila.get(i).pop();//O(1)
                }
            }
            if(loctB){ //c6
                laPila.get(i).push(a); //O(1)
            }
        }
    }
}
```

T(n)= c+ n*c2*(m*c5)
O(n)= c+ n*c2*(m*c5)
O(n)= n*c2*(m*c5) R.S.
O(n)= n*c2*(m) R.P.
O(n)= n*m R.P.

```
private void pileOnto(int a, int b){
    int pA=0; //c1
    int pB=0; //c2
    if (a!=b){ //c3
        for (int i = 0; i < tam; i++) {// n*c4
            boolean loctA = laPila.get(i).contains(a); //O(1)
            boolean loctB = laPila.get(i).contains(b); //O(1)

            if(loctA){ //c5
                pA=i; //c6
            }

            if(loctB){ //c7
                pB=i; //c8
            }
        }
        if(pA!=pB){ //c9
            int pX=(int)laPila.get(pB).pop(); //O(1)
            while(pX!=b){ //m*c10
                laPila.get(pX).push(pX); //O(1)
```

```
            pX=(int)laPila.get(pB).pop();//O(1)
        }

        //para los elementos que estaban apilados en a
        Stack elemtA= new Stack(); //c11
        pX=(int)laPila.get(pA).pop(); //O(1)
        while(pX!=a){ //m*c12
            elemtA.push(pX);//O(1)
            pX=(int)laPila.get(pA).pop(); //O(1)
        }
        elemtA.push(a); //O(1)

        //paso a b los elementos de la pila elemtA en el orden original
        while(!elemtA.empty()){//m*c13
            laPila.get(pB).push(elemtA.pop()); //O(1)
        }
      }
    }
  }
```

T(n)= c+ n*c4*(m*c10+m*c12+m*c13)
T(n)= c+ n*c4*(m*c10+m*c12+m*c13)
O(n)= n*c4*(m*c10+m*c12+m*c13) R.S.
O(n)= n*c4*(m*(c10+c12+c13))
O(n)= n*c4*(m) R.P.
O(n)= n*m R.P.

```
  private void pileOver (int a, int b){
    int pA=0; //c1
    int pB=0; //c2
    if (a!=b){ //c3
      for (int i = 0; i < tam; i++) { //n*c4
        boolean loctA = laPila.get(i).contains(a); //O(1)
        boolean loctB = laPila.get(i).contains(b); //O(1)

        if(loctA){ //c5
          pA=i; //c6
        }

        if(loctB){ //c7
```

```
                pB=i; //c8
            }
        }
        if(pA!=pB){ //c9
            //para los elementos que estaban apilados en a
            Stack elemtA= new Stack(); //c9
            int pX=(int)laPila.get(pA).pop(); //O(1)
            while(pX!=a){ //m*c10
                elemtA.push(pX); //O(1)
                pX=(int)laPila.get(pA).pop(); //O(1)
            }
            elemtA.push(a); //O(1)
            while(!elemtA.empty()){ // m*c11
                laPila.get(pB).push(elemtA.pop()); //O(1)
            }
        }
    }
}
```

T(n)= c+ n*c4*(m*c10+m*c11)
T(n)= c+ n*c4*(m*c10+m*c11)
O(n)= n*c4*(m*c10+m*c11) R.S.
O(n)= n*c4*(m*(c10+c11))
O(n)= n*c4*(m) R.P.
O(n)= n*m R.P.

```
    private void imprimir(){
        Stack pilaNueva = new Stack(); //c1
        for (int i = 0; i < tam; i++) { //n*c2
            System.out.print(i+":"); //c3
            while (!laPila.get(i).empty()) { //m*c4
                pilaNueva.push(laPila.get(i).pop()); //O(1)
            }
            while(!pilaNueva.empty()) //m*c5
            {
                System.out.print(" "+pilaNueva.pop()); //O(1)
            }
            System.out.println(); //c6
        }
    }
}
```

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

T(n)= c+ n*c2*(m*c4+m*c5)
T(n)= c+ n*c2*(m*c4+m*c5)
O(n)= T(n)= n*c2*(m*c4+m*c5)) R.S.
O(n)= n*c2*(m*(c4+c5))
O(n)= n*c2*(m) R.P.
O(n)= n*m R.P.

4. Las variables n y m usadas en el cálculo de las complejidades éstas representan el tamaño de la entrada que se da en la entrada del usuario al algoritmo (n el tamaño del ArrayList que contendrá las pilas y m la cantidad de elementos hallados en la pila) y en función de la cual se mide la eficiencia de éste, pues permite ver la cantidad de tiempo que le tomará realizar cierta cantidad de pasos para emitir la respuesta correspondiente a la entrada.
5.

4) Simulacro de Parcial

1. 3. lista.size()>0 7. Lista.add(auxiliar.pop())
2. *A) 12. auxiliar1.size() > 0, 16. auxiliar2.size() > 0. B) personas.offer(edad)*
3. C) O(n^2)