Laboratorio No. 2: Linked List y Array List

Cristian Camilo Rendón Cardona
Universidad Eafit
Medellín, Colombia
crendo11@eafit.edu.co

Jhesid Steven Suerez Berrio
Universidad Eafit
Medellín, Colombia
jssuarezb@eafit.edu.co

1) Ejercicios para entregar en GitHub

1.1- Para la solución de este problema se recurre a un llamado recursivo donde se va eliminando el primer elemento de la lista (T(n-1)).

```
static int multiply(LinkedList<Integer> a){
        int m=0;
        int mult=proceso(a,m);

        return mult;
}

static int proceso(LinkedList<Integer> a,int m){
    if(a.isEmpty()!=true){
        m=a.pop()*proceso(a,m);     \\ T(n-1)
    }
    return m;
}
```

1.2- Para la solución a este problema se hace el uso de un ciclo que recorre la lista para verificar que el elemento a insertar no se encuentra. En caso de que no se encuentre, se agrega el elemento al inicio de la lista, pero si se el elemento ya existe en la lista el para el ciclo y no se agrega de nuevo. Debido al uso del método get() (O(n)) dentro del ciclo, la complejidad asintótica de la función se vuelve O(n^2).

```
static void smartInsert(LinkedList<Integer> a, int n){
        for (int i = 0; i < a.size(); i++) {      \\n
            int z=(int)a.get(i);                  \\n*n
            if (z==n) {
                break;
            }
            if ((z!=n)&&(i==a.size()-1)) {
                a.add(n);
            }
        }

}
```

1.3- En la solución al problema se crean dos ciclos anidados donde el primero mueve el pivote por la lista y el segundo hace l suma por la derecha y la izquierda del pivote para encontrar el balance adecuado. Debido al uso de get() dentro del ciclo anidado, la complejidad se eleva hasta O(n^3).
Debido a la longitud del código solo se presentan las líneas relevantes a la complejidad.

```
for (int j = i + 1;j < a.size();j++) {        \\n

    ...

    for (int j = 0; j < i;j++) {              \\n*n
        sumi = sumi + a.get(j);               \\n*n*n
    }

}
```

1.4- Se hace uso de las lista para almacenar las neveras conforme se van fabricando, así la primer nevera en fabricarse será la última en la lista.
1.5- Se trabaja en la plantilla de la clase LinkedListMauricio para la implementación del método. Se crea un lista simplemente enlazada.
1.6- Las pruebas aplicadas a las listas enlazadas se pueden encontrar en el main del código.
1.7- Para el ejercicio del banco se crea un ciclo que saca un elemento de cada fila y lo guarda en la lista del cajero 1 o el 2 según corresponda con la ayuda de una variable booleana que indica a cuál de ellos le toca atender.

3) Simulacro de preguntas de sustentación de Proyectos

1. A continuación se presenta la tabla de complejidades para os ejercicios del numeral 1

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

	ArrayList	LinkedList
Ejercicio 1.1	*O(n)*	*O(n)*
Ejercicio 1.2	*O(n^2)*	*O(n)*
Ejercicio 1.3	*O(n^3)*	*O(n^2)*
Ejercicio 1.4		

2. **…**
3. **…**
4. **…**
5. ***n es el número de bloques a mover y m es el número de operaciones dictadas por el usuario.***

4) Simulacro de Parcial

1. *c*
2. *b*
3. *a). q.size() > 0*
   b). <=
   c). q.add(q.remove)
   d). q.remove()
4. *a). Lista.size()*
   b). Lista.push(auxiliary.removeFirst())
5. *a). 12: auxiliar1.size() > 0 16: auxiliar2.size() > 0*
   b). personas.offer(mayorEdad)
6. *c*