Laboratorio Nro. 3
Escribir el tema del laboratorio

Simón Correa Henao
Universidad Eafit
Medellín, Colombia
scorreah@eafit.edu.co

David Gómez Correa
Universidad Eafit
Medellín, Colombia
dgomezc10@eafit.edu.co

3) Simulacro de Preguntas Sustentación de Proyecto

3.1

Complejidad con ArrayList:

Si se inserta cada elemento al inicio:

O(n * n!)

Si se inserta cada elemento al final:

O(n^2)

Complejidad con LinkedList:

O(n)

Para representar el mapa de Medellín con matrices, la complejidad en memoria sería de:

O(n)

La forma de solucionar el problema de que los identificadores no empiecen en 0, y además se salten números, es sencillamente manejar por separado el Id de cada ubicación, de la instancia en la que queda almacenada dentro del *arrayList*. Siendo este el caso, se guardaría como un atributo del tipo de dato de la ubicación el Id que le pertenece.
En otras palabras, la forma de solucionarlo que utilizamos fue crear dos nuevos tipos de datos, uno para los vértices, y uno para los arcos, y que ambos tipos heredaran de una clase abstrapta de tipo lugar. Esto con el objetivo de hacer un solo ArrayList de tipo lugar, que pueda contener tanto Vertices, como Arcos. Y dentro de cada uno de estos tipos de datos colocamos el Id como otro atributo más, para no preocuparnos por él. Así a la hora de buscar un Lugar específico, se haría la búsqueda del Id que coincide.

3.3 y 3.4

La complejidad del algoritmo del Teclado Roto es O(n)

```
public static void teclado(String a){
        LinkedList<Character> texto = new LinkedList<>();
        LinkedList<Character> aux = new LinkedList<>();

        boolean inicio = false;
        for (int i = 0; i < a.length(); i++) {      //O(n)
```

```
            if (a.substring(i, i+1).equals("[")) {
                if(aux.size() != 0){
                    texto.addAll(0, aux);
                    aux.clear();
                }
                inicio = true;
                continue;
            }
            else if (a.substring(i, i+1).equals("]")) {
                inicio = false;
                continue;
            }
            else if (inicio)    aux.addLast(a.charAt(i));
            else if (!inicio)   texto.addLast(a.charAt(i));
        }

        if (aux.size() != 0) texto.addAll(0, aux);
        aux.clear();
        Character[] txtRoto = new Character[texto.size()];
        txtRoto = texto.toArray(new Character[texto.size()]);
        for (int i = 0; i < txtRoto.length; i++) {      //O(n)
            System.out.print(txtRoto[i]);
        }
        System.out.println("");

    }

}
```

Siendo n la longitud del *String* que recibe la función por parametro.
Esto se da, considerando que en el algoritmo primero se utiliza un ciclo *for*, donde se llena correctamente la lista con cada carácter del *String* recibido, y cuya complejidad en el peor de los casos es lineal (O(n)). Y luego, la lista, ya con el resultado final de cómo se desea exportar en la salida, se convierte en un Array, cuya operación de acceso de complejidad O(1), para imprimirlo por medio de un ciclo *for.*
Resumiendo, en el peor de los casos, la complejidad de almacenar correctamente los elementos en la lista sería de O(n), y la complejidad de imprimir el arreglo con los elementos que contenían la lista previamente utilizada, también es O(n). Por lo que finalmente la complejidad del algoritmo completo terminaría siendo O(n).

4) Simulacro de Parcial

Respuestas:

4.1 B) y A)
4.2 C)
4.4 stack.pop() 4.4.1 A)
4.5 A)

4.6 B)
4.8 D)
4.8.1) B
4.8.2) C
4.8.3) C
4.9.1) D
4.9.2) A
4.9.3) B
4.10.1) B
4.10.2) D
4.12 s1.size()-1 s1.pop() s2
4.12.1 IV) 4.12.2 A)
4.13.1 C) 4.13.2 A)