Actividad 1 – Estructuras de datos no lineales

Complejidad algorítmica

1. Para cada uno de los tiempos que toma un algoritmo en terminar, demostrar que cada uno es de la complejidad dada:
   a. $T_A(n) = 2n^3 - 3n^2 + 1$ es $O(n^3)$
   Hay 3 expresiones en la formula, 2 exponenciales y una constante, la constante es eliminada, por lo que nos quedan 2 exponenciales, la complejidad del algoritmo seria $O(n^3)$ debido a que 3 es el mayor exponente entre las 2 expresiones restantes.

   b. $T_A(n) = n^5 + 4^2 - \sqrt{n} + 1$ es $O(n^5)$
   Hay 4 expresiones en la formula, 2 exponenciales y 2 constantes, simplificaremos para verlo más fácilmente:
   $$T_A(n) = n^5 + 16 - n^{\frac{1}{2}} + 1$$
   Teniendo en cuenta que 5 es el mayor exponente de la formula podemos decir que el algoritmo es de complejidad:
   $$O(n\text{^}5)$$

   c. $T_A(n) = n^2 \log n + 2n^4 + \sqrt{2}n$ es $O(n^4)$
   Hay 3 expresiones en la formula, simplificando las constantes de 2 de ellas tendríamos:
   $$T_A(n) = n^2 \log n + n^4 + n$$
   Por lo que tenemos:
   $$O(n^2 \log(n)) + O(n^4) + n$$
   Teniendo en cuenta la jerarquía de crecimiento la complejidad seria de:
   $$O(n^4)$$

2. Calcule paso a paso la complejidad del siguiente algoritmo:

```
void XXXXXX(int n){
    int x = 0;

    for (int i = 1; i <= n; i *= 5) {
        int j = 1;
        for (int j = 1; j <= n; j += 2) {
            x = x + j;
        }
        for (int k = n; k >= 1; k /= 2) {
             x = x + 1;
        }
    }
}
```

a. Operaciones básicas:
   Cada uno de los bucles tiene 1 asignación, 1 comparación y una operación matemática por cada uno de sus ciclos.
   Además de esto, el ciclo externo tiene una asignación adicional y cada uno de los ciclos internos tiene una asignación y una operación adicional.

b. Variables que afectan al número de operaciones:
   i. n

c. Análisis de estructuras de control:
   i. Bucle externo:
      Este bucle comienza en $i = 1$ y en cada iteración se multiplica i por 5, este crecimiento es logarítmico en base 5, esto nos dice que el número de iteraciones del bucle será $\log_5(n)$, por lo que su complejidad seria de:
      $$O(\log(n))$$
   ii. Bucle interno #1:
      Este bucle comienza en $j = 1$ y en cada iteración se aumenta j en 2, por lo que básicamente se ejecuta $\frac{n}{2}$ veces, lo que equivale a una complejidad de:
      $$O(n)$$
   iii. Bucle interno #2:
      Este bucle comienza en $k = n$ y en cada iteración k se divide en 2, este crecimiento es logarítmico en base 2, esto nos dice que el número de operaciones del bucle será de $\log_2(n)$, por lo que su complejidad será de:
      $$O(\log(n))$$

d. Expresión:

$$T(n) = O(\log(n)) \cdot \big(O(n) + O(\log(n))\big)$$

$$Desarrollamos\ el\ parentesis$$

$$O(n) > O(\log(n))$$

$$Entonces$$

$$T(n) = O(\log(n)) \cdot O(n)$$

$$T(n) = O(\mathrm{n} \cdot \log(n))$$

3. Suponga que se tienen los siguientes dos algoritmos para resolver el mismo problema, y suponga que la función **buscar** tiene complejidad $O(n \cdot \log n)$.

```
boolean Ex1(int[] a, int elem) {
    int pos = buscar(a, elem);
    int n = a.length;
    int x = pos;

    for (int i = 0; i < n; ++i) {
        x += 2;
        for (int j = 0; j< n; ++j) {
            if (a[j] > a[pos]) {
                x++;
            }
        }
    }
    return x > elem;
}
```

```
boolean Ex2(int[] a, int elem) {
    int n = a.length;
    int x = 0;

    for (int i = 0; i < n; ++i) {
        int pos = buscar(a, elem);
        x += pos + 2;
        for (int j = 0; j < n; ++j) {
            if (a[j] > a[pos]) {
                x++;
            }
        }
    }
    return x > elem;
}
```

Indique cuál de los dos algoritmos se queda para resolver el problema. Justifique muy bien su respuesta.

a. Complejidad del algoritmo Ex1:
   i. Operaciones básicas:
      - 2 externas
      - Cada bucle tiene 1 asignación 1 comparación y operación matemática
      - Internamente el bucle externo tiene una operación
      - Internamente el bucle interno tiene 1 comparación y una asignación.
   ii. Variables que afectan el número de operaciones:
      - n

iii. Análisis de estructuras de control:

- Bucle externo:
  Este bucle comienza con i=0 y en cada iteración se suma 1 a i, este es un crecimiento lineal por lo que este bucle se ejecuta n veces, por lo que su complejidad será de:
  $$O(n)$$
- Bucle interno:
  Este bucle comienza con j=0 y en cada iteración se suma 1 a j, este es un crecimiento lineal, por lo que este bucle se ejecuta n veces, por lo que su complejidad será de:
  $$O(n)$$

iv. Expresión:

$$T(n) = O(n \cdot \log(n)) + O(n) \cdot O(n)$$

$$T(n) = O(n \cdot \log(n)) + O(n^2)$$

Esto al ser una suma, solo nos queda el de mayor jerarquia, por lo tanto:

$$T(n) = O(n^2)$$

b. Complejidad del algoritmo Ex2:

i. Operaciones básicas:

- 2 externas
- Cada bucle tiene 1 asignación 1 comparación y operación matemática
- Internamente el bucle externo tiene 2 operaciones más, y además de esto utiliza la función buscar, cuya complejidad es $O(n \cdot \log(n))$
- Internamente el bucle interno tiene 1 comparación y 1 asignación.

ii. Variables que afectan el número de operaciones:

- n

iii. Análisis de estructuras de control:

- Bucle externo:
  Este bucle comienza con $i = 0$ y en cada iteración se aumenta i en 1, este es un crecimiento lineal, por lo que este bucle se ejecuta n veces lo que nos da una complejidad de:
  $$O(n)$$

- Bucle interno:
  Este bucle comienza con j=0 y en cada iteración se suma 1 a j, este es un crecimiento lineal, por lo que este bucle se ejecuta n veces, por lo que su complejidad será de:

$$O(n)$$

iv. Expresión:

$$T(n) = O(n) \cdot \big(O(n \cdot \log(n)) + O(n)\big)$$

$$T(n) = O(n) \cdot O(n \cdot \log(n)) + O(n) \cdot O(n)$$

$$T(n) = O(n^2 \cdot \log(n)) + O(n^2)$$

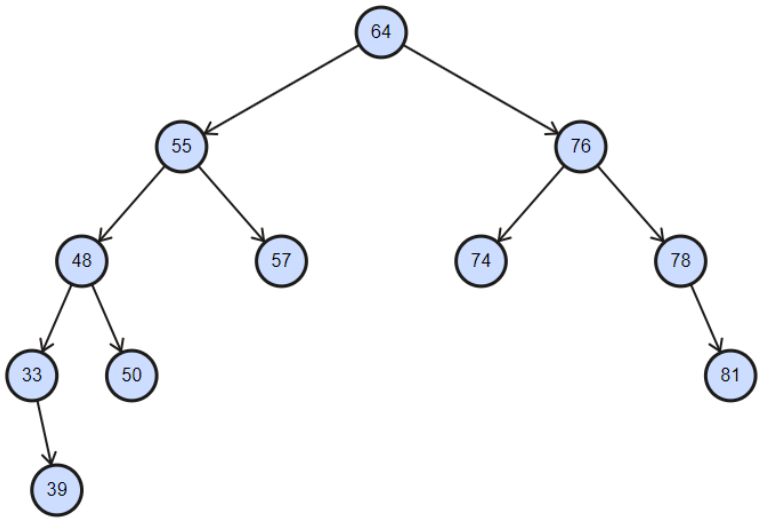
$$T(n) = O(n^2 \cdot \log(n))$$

Por lo tanto, el algoritmo elegido sería el Ex1, ya que:

$$O(n^2) < O(n^2 \cdot \log(n))$$

Arboles Binarios

1. Para el siguiente árbol binario:



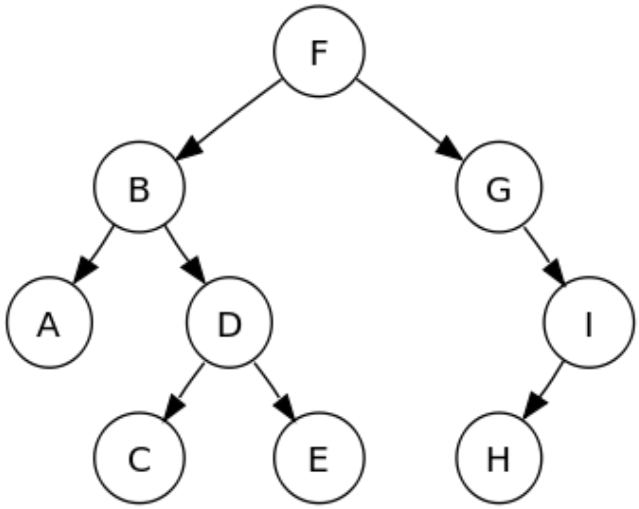
   a. Peso: 11
   b. Altura: 4
   c. Hojas: 5
   d. Una rama:
      - 64-55-48-33-39
      - 64-55-48-50
      - 64-55-57
      - 64-76-74
      - 64-76-78-81
   e. Recorrido en in-orden

      $33 -> 39 -> 48 -> 50 -> 55 -> 57 -> 64 -> 74 -> 76 -> 78 -> 81$
   f. Recorrido en pre-orden

      $64 -> 55 -> 48 -> 33 -> 39 -> 50 -> 57 -> 76 -> 74 -> 78 -> 81$
   g. Recorrido en post-orden

      $39 -> 33 -> 50 -> 48 -> 57 -> 55 -> 74 -> 81 -> 78 -> 76 -> 64$

2. Para el siguiente árbol binario:



a. Altura: 3
b. Número de niveles: 4
c. Ancestro común de la E y la A: B
d. Peso del árbol izquierdo de la F: 9
e. Recorrido en in-orden

$$A -> B -> C -> D -> E -> F -> G -> H -> I$$

f. Recorrido en pre-orden

$$F -> B -> A -> D -> C -> E -> G -> I -> H$$

g. Recorrido en post-orden

$$A -> C -> E -> D -> B -> H -> I -> G -> F$$

h. Recorrido por niveles

$$F -> B -> G -> A -> D -> I -> C -> E -> H$$

i. Hojas: 4

3. Reconstruya el árbol binario que posee los siguientes recorridos:

Pre-orden: 1 – 2 – 3 – 4 – 5 – 6 – 7

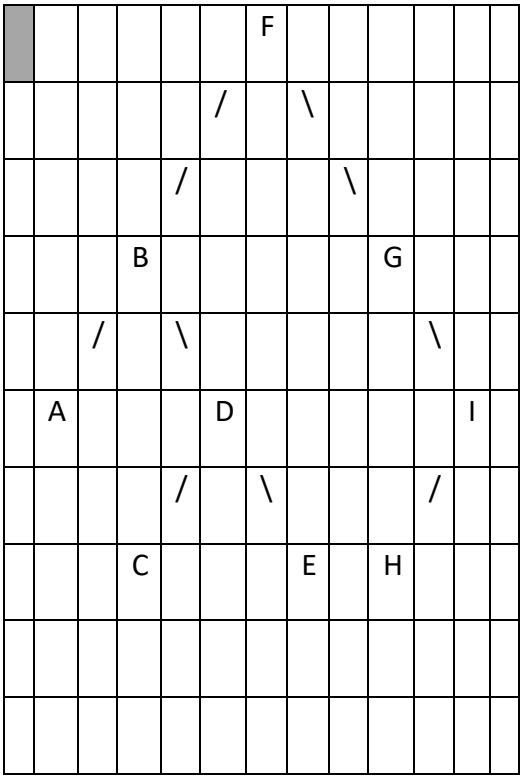
In-orden: 3 – 2 – 5 – 4 – 1 – 6 - 7

				1				
			/		\			
		2				6		
	/		\				\	
3				4				7
			/					
		5						

4. Reconstruya el árbol binario que posee los siguientes recorridos:

Post-orden:
A – C – E – D – B – H – I – G – F

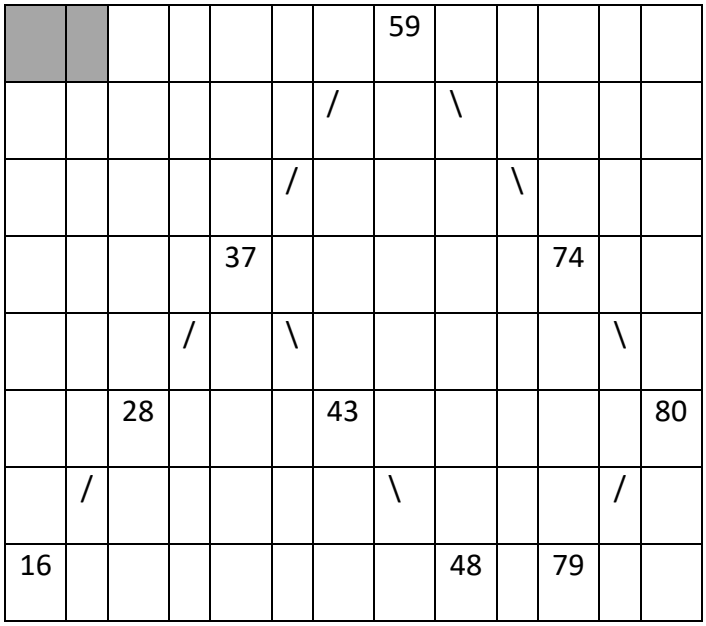
In-orden: A – B – C – D – E – F – G – H – I



5. Reconstruya el árbol binario que posee los siguientes recorridos:

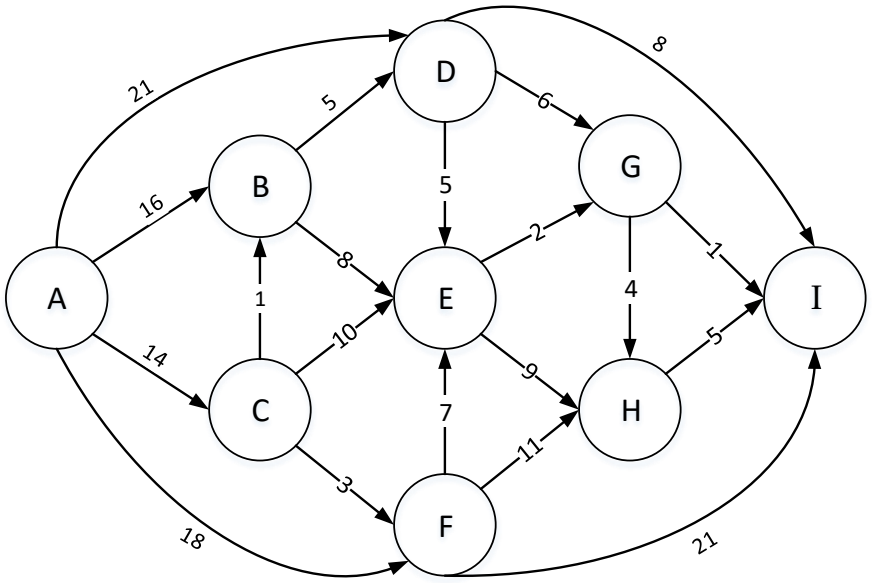
Pre-orden: 59 – 37 – 28 – 16 – 43 – 48 – 74 – 80 – 79

In-orden: 16 – 28 – 37 – 43 – 48 – 59 – 74 – 79 – 80



Grafos - Algoritmo de Dijkstra

Para el siguiente grafo:



a. Encuentre (paso a paso) los costos mínimos de los caminos que parten del vértice **A** usando el algoritmo de Dijkstra

Iteración 1:

Vértice	**Seleccionados**		***A***	***B***	***C***	***D***	***E***	***F***	***G***	***H***	***I***
A	C	Costo:	-	16	14	21	∞	18	∞	∞	∞
		Anterior:	0	∞	∞	∞	∞	∞	∞	∞	∞

Iteración 2:

Vértice	**Seleccionados**		***A***	***B***	***C***	***D***	***E***	***F***	***G***	***H***	***I***
C	B	Costo:	0	15	-	21	24	17	∞	∞	∞
		Anterior:	0	16	14	21	∞	18	∞	∞	∞

Iteración 3:

Vértice	**Seleccionados**		***A***	***B***	***C***	***D***	***E***	***F***	***G***	***H***	***I***
B	F	Costo:	0	-	14	20	23	17	∞	∞	∞
		Anterior:	0	15	14	21	24	17	∞	∞	∞

Iteración 4:

Vértice	Seleccionados		*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*
F	D	Costo	0	15	14	20	23	-	∞	28	38
		Anterior:	0	15	14	20	23	17	∞	∞	∞

Iteración 5:

Vértice	Seleccionados		*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*
D	E	Costo	0	15	14	-	23	17	26	28	28
		Anterior:	0	15	14	20	23	17	∞	28	38

Iteración 6:

Vértice	Seleccionados		*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*
E	G	Costo	0	15	14	20	-	17	25	28	28
		Anterior:	0	15	14	20	23	17	26	28	28

Iteración 7:

Vértice	Seleccionados		*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*
G	I	Costo	0	15	14	20	23	17	-	28	26
		Anterior:	0	15	14	20	23	17	25	28	28

Iteración 8:

Vértice	Seleccionados		*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*
I	FIN	Costo	0	15	14	20	23	17	25	28	-
		Anterior:	0	15	14	20	23	17	25	28	26

b. ¿Cuál es el camino entre el vértice A y el vértice I y cuánto es el costo de tal camino, de acuerdo con el algoritmo de Dijkstra?
El camino sería:

$$A \to C \to B \to E \to G \to I$$

Con un costo total de 26.

Actividad 2 – Estructuras de datos lineales enlazadas – Central de Pacientes

URL con el código fuente

https://github.com/svianel04744/desarrollo-de-software/tree/main/Gu%C3%ADa%203/A2%20-%20Central%20de%20Pacientes

Descripción del programa

El programa cuenta con una interfaz de usuario desarrollada con Java Swing, la cual nos permite agregar un nuevo paciente, buscar un paciente dada su identificación única o eliminar un paciente dada su identificación única.

- Agregar paciente:
    - Ingrese los siguientes datos:
        - Identificación
        - Nombre
        - Edad (Numero)
        - Clínica
    - Oprima el botón “Agregar”
- Buscar paciente:
    - Ingrese la identificación del paciente que quiere buscar
    - Oprima el botón “Buscar”
    - *NOTA: Si desea limpiar el filtro de búsqueda, limpie el campo de búsqueda y oprima el botón “Buscar”*
- Eliminar paciente:
    - Ingrese la identificación del paciente que quiere eliminar
    - Oprima el botón “Eliminar”

Uso y ejecución

En la URL anteriormente entregada, se puede descargar el código fuente únicamente las carpetas básicas con las clases JAVA del proyecto, esto permite la facilidad de ser importado al IDE de su preferencia.

Para la ejecución diríjase a la ubicación del proyecto en su sistema y ejecute el archivo `CentralPacientes.jar`.