1. Amplía la función **`construirArbolAnalisis`** para que pueda manejar expresiones matemáticas que no tienen espacios entre cada carácter.

2. Modifica las funciones **`construirArbolAnalisis`** y **`evaluar`** para que puedan manejar las sentencias booleanas (**`and`**, **`or`**, y **`not`**). Recuerda que "**`not`**" es un operador unario, por lo que esto complicará un poco la realización del código.

3. Utilizando el método **`encontrarSucesor`**, escribe un recorrido **`inorden`** **no recursivo** para un árbol binario de búsqueda.

4. Modifica la implementación de árbol binario de búsqueda para que maneje correctamente claves duplicadas. Es decir, si una clave ya está en el árbol, entonces la nueva carga útil debería sustituir a la antigua en lugar de **`agregar`** otro nodo con la misma clave.

5. Modifica la función **`imprimirExpresion`** para que no incluya un par de paréntesis 'extra' alrededor de cada número.

6. Utilizando el método **`construirMonticulo`**, escribe una función de ordenamiento que pueda ordenar una lista en tiempo $\boldsymbol{O(nlogn)}$. (Buscar heapsort)

7. Implementa un montículo binario como un montículo **`máx`**.

8. Utilizando la clase **`MonticuloBinario`**, implementa una nueva clase llamada **`ColaPrioridad`**. Esta nueva clase **`ColaPrioridad`** debe implementar el constructor, además de los métodos **`agregar`** y **`avanzar`**.

---

11. Crea el grafo correspondiente a la siguiente lista de aristas.

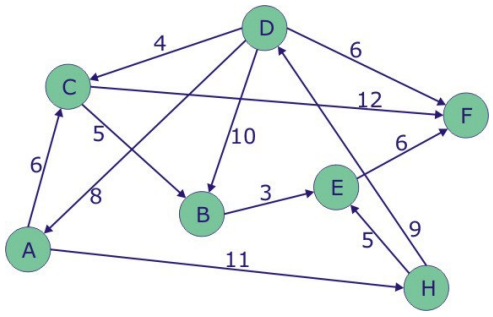
de	a	costo
1	2	10
1	3	15
1	6	5
2	3	7
3	4	7
3	6	10
4	5	7
6	4	5
5	6	13

12. Haciendo caso omiso de las ponderaciones, realiza una **búsqueda en anchura** en el grafo de la pregunta anterior.

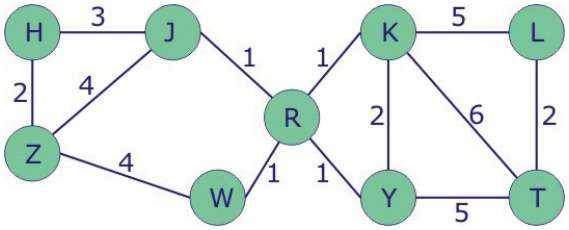
13. ¿Cuál es el tiempo de ejecución **O-grande** de la función **`construirGrafo`**?

14. Utilizando el ejercicio 8, implementar el método **`decrementarClave`** en la clase **`MonticuloBinario`** para que funcione el algoritmo de **Dijkstra**. Muestra cada paso al aplicar el algoritmo de **Dijkstra** al grafo resultante del problema 11.

15. Dado el grafo dirigido y valorado de la figura, encuentra el camino más corto desde el vértice **A** a todos los demás vértices usando el algoritmo de **Dijkstra**. Describe el proceso paso a paso.



16. Usando el algoritmo de **Prim**, encuentra el árbol de expansión de ponderación mínima
17. ¿Cuál es el tiempo de ejecución **O-grande** para el algoritmo de **Prim** del árbol de expansión mínimo?
18. Dado el grafo de la figura, encuentra un árbol de expansión de coste mínimo describiendo el proceso paso a paso mediante el algoritmo de **Prim**



19. Escribe el método `transponer` para la clase `Grafo`.
20. Utilizando la **búsqueda en anchura**, escribe un algoritmo que puede determinar la ruta más corta de cada vértice a cada uno de los otros vértices. Esto se llama el ***problema de la ruta más corta de todas las parejas.*** **(Algoritmo de Floyd- Warshall)**

Problema 1.- Dado el **TAD ArbolBinario** con las especificaciones detalladas en los apuntes de la asignatura y con una implementación mediante **nodos** y **referencias**, se pide:

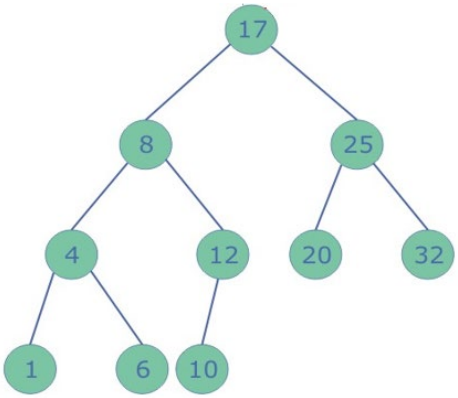
a) Implementar las siguientes operaciones manejando árboles binarios de **enteros**.

- **numNodos(ArbolBinario) devuelve (entero)**
  - **efecto:** Devuelve la cantidad de nodos de un árbol binario**.**
- **numHojas(ArbolBinario) devuelve (entero)**
  - **efecto:** Devuelve el número de hojas de un árbol binario.
- **profundidad(ArbolBinario) devuelve (entero)**
  - **efecto:** Devuelve la profundidad de un árbol binario.

b) Implementar la operación **Espejo**, que dado un árbol binario **A** creará otro que sea su imagen especular, es decir, su simétrico.

- **Espejo(ArbolBinario) devuelve (ArbolBinario)**
  - **efecto:** Devuelve el árbol binario **Espejo** del árbol binario de partida.

Problema 2.- Implementar un programa en **C++ que realice todos los recorridos (preorden, inorden, postorden y recorrido en anchura)** del árbol siguiente:



Problema 3.- Construye un árbol binario de búsqueda (**ABB**) a partir de la siguiente lista:

G B Q A C K F P D E R H

Para el árbol obtenido, muestra la evolución de los siguientes algoritmos de recorrido de árboles binarios:

a) Recorrido en **preorden**
b) Recorrido en **inorden**
c) Recorrido en **postorden**
d) Recorrido en **anchura**

Escribe el código en **Python** y en **C++** de los algoritmos **recursivos** apropiados para determinar:

a) La cantidad de nodos de un árbol binario de búsqueda
b) El número de hojas de un árbol binario de búsqueda
c) La profundidad (altura) de un árbol binario de búsqueda