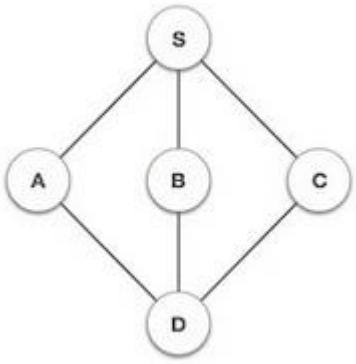
Práctica: Grafos con listas de adyacencia.
16 y 18 de noviembre – Estructura de Datos

En esta práctica crearemos la estructura de datos Grafos con listas de adyacencia. La idea de esta práctica es desarrollar esta implementación y que al menos pueda realizar recorridos en anchura y en profundidad. Se te sugiere una estructura de clases basada en la implementación de Weiss, sin embargo, puedes tu seguir tu propia implementación.

En la implementación sugerida se utilizan las siguientes estructuras de datos; HashMap, LinkedList, Queue.

Antes de comenzar, determina cómo serían los recorridos en anchura y profundidad del siguiente grafo:



1. Se generarán la clase Arista, Vértice y Grafo.
2. Crea la clase Arista;
   a. Variables de instancia que puedes dejar públicas o agregar los correspondientes get y set para cada una.
      Vertice destino ---- indica a dónde lleva la arista
      double costo ----- indica el peso de la arista, si es un grafo ponderado.
   b. Métodos
      - Constructor que recibe Vertice v, double costo, el cual inicializa las variables de instancia.
3. Crea la clase Vértice;
   c. Variables de instancia
      String nombre ---- nombre del vértice
      List<Arista> adyacentes ---- lista de los vértices que son adyacentes a éste.
      double distancia
      int marcado
      Vertice anterior
   d. Métodos
      - Constructor que recibe el nombre e inicializa las variables de instancia, creando la lista de adyacencia (vacia).

- void reinicia() --- este método lo que hace es establecer el atributo marcado en 0, anterior a null y distancia a Grafo.INFINITO (constante de la clase grafo)

4. Crea la clase Grafo;
   e. Variables de instancia
      Map<String, Vertice> mapaVertices;
   f. Constante
      public static final double INFINITO=Double.MAX_VALUE;
   g. Métodos
      - Constructor que inicializa la variable de instancia mapaVertices=new HashMap<String,Vertice>();
      - private Vertice getVertice(String nombre) --- Este método verificará si existe en el diccionario el vértice a través de su nombre, en caso de que no, se crea el vértice y se agrega al diccionario, devuelve el vértice creado o encontrado en el mapa.
      - public void agregaArista(String origen, String destino, double costo) --- A través de getVertice y con los strings nombre y destino, inicializamos dos variables tipo Vertice (v y w). Una vez identificados los vértices, agregamos a la lista de adyacencia del vértice v, una arista al destino w y el costo.
      - public void reiniciaTodos() ---Este método reinicia todos los vértices del grafo. Este método te sirve para reiniciar a las condiciones iniciales, todos los vértices del grafo antes de los recorridos.
      - public String recorridoEnAnchura(String origen) --- Debe devolver el String que contiene el recorrido en anchura del grafo, empezando por el vértice origen. Sigue el pseudocódigo de la presentación de clase.
      - public String recorridoEnProfundidad(String origen)--- Debe devolver el String que contiene el recorrido en profundidad del grafo, empezando por el vértice origen. Sigue el pseudocódigo de la presentación de clase.

      -

2. Prueba tu clase,

Referencias:

- Méndez, A. **Graphs** – Presentación de clase.
- Weiss, M.A. (2012) **Data Structures and Algorithm Analysis in Java**. US:Pearson.