UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA

FACULTAD DE INGENIERIA DE PRODUCCION Y SERVICIOS

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

Estudiante:

Chávez Cáceres, Samir Diego

CUI: 20200611

Curso: Laboratorio Estructuras de Datos y Algoritmos

Docente:

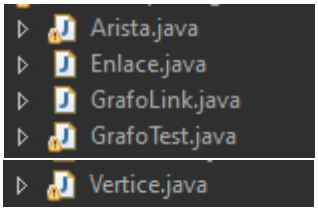
Edith Pamela Rivero Tupac de Lozano

Arequipa - Perú

Julio 2021

2. Implementar el cogido de Grafo cuya representacion sea realizada mediante LISTA DE ADYACENCIA. (3 puntos)

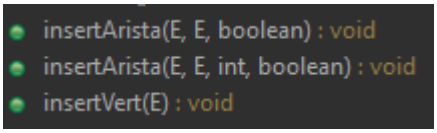
Se trabajara con las siguientes clases:



Empezando por GrafoLink, este contiene una lista enlazada de vértices.

```java
public class GrafoLink<E> {
    protected ListEnlazada<Vertice<E>> listVert;

    public GrafoLink() {
        listVert = new ListEnlazada<Vertice<E>>();
    }
```

El cual contara con los métodos:




```java
    public void insertVert(E data) {
        Vertice<E> nuevo = new Vertice<E>(data);
        if(this.listVert.contiene(nuevo) != null) {
            System.out.println("Vertice ya insertado");
            return;
        }
        this.listVert.agregar(nuevo);
    }

    public void insertArista(E verOrig, E verDest, boolean dirig) {
        insertArista(verOrig, verDest, -1, dirig);
    }

    public void insertArista(E verOrig, E verDest, int weight, boolean dirig) {
        Vertice<E> refOrig = this.listVert.contiene(new Vertice<E>(verOrig)).valor;
        Vertice<E> refDest = this.listVert.contiene(new Vertice<E>(verDest)).valor;

        if(refOrig == null || refDest == null) {
            System.out.println("Vertice origen / dest no existe");
            return;
        }
        if(refOrig.listArt.contiene(new Arista<E>(refDest)) != null) {
            System.out.println("Arista ya insertada");
            return;
        }
        if(dirig) { //si el enlace sera dirigido o no
            refOrig.listArt.agregar(new Arista<E>(refDest, weight));
        } else {
            refOrig.listArt.agregar(new Arista<E>(refDest, weight));
            refDest.listArt.agregar(new Arista<E>(refOrig, weight));
        }
    }

```

InsertVertice nos inserta en nuestra lista enlazada un objeto vértice, el cual será colocado como la nueva cabeza de la lista, por otro lado insertArista() dependiendo del parámetro booleano dirig especificaremos si la conexión será dirigida o no.

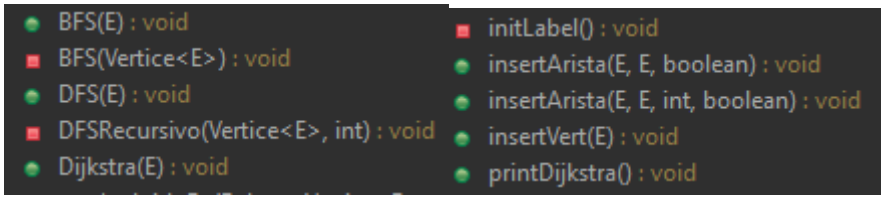
La clase arista representara a la ya mencionada, asi que contara con un peso, una etiqueta para los algoritmos y asi como una referencia al vertice conectado.

La clase vértice contendrá el dato a guardar, una etiqueta para los algoritmos, una referencia a un vértice adyacente para los algoritmos de recorrido minimo, y una lista enlazada de aristas que partan de nuestro vértice.

La clase lista enlazada, trabajara con enlaces los cuales contienen el elemento y la referencia al siguiente elemento.

3. Implementar BSF, DFS y Dijkstra con sus respectivos casos de prueba. (5 puntos)

A nuestra clase GrafoLink se le agregan los siguientes métodos:



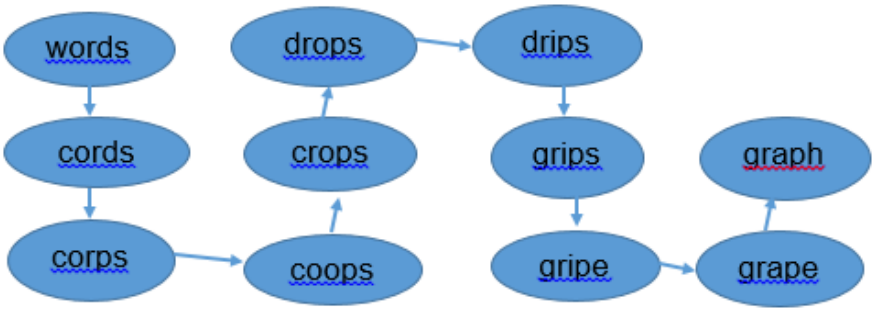
De acuerdo a las definiciones teóricas y al pseudocodigo brindado en clases de teoría se muestran la parte más importante de cada algoritmo.

```
94    private void BFS(Vertice<E> v) {
95        QueueLink<Vertice<E>> seq = new QueueLink<>();
96        seq.enqueue(v);
97        v.label = 1;
98        int i = 0;
99        while(!seq.isEmpty()) {
100           System.out.print("Vert\t --> Enlaces\n");
101           System.out.println("Gen "+i+":\n"+seq+"\n");
102           QueueLink<Vertice<E>> sigSeq = new QueueLink<Vertice<E>>();
103           while(!seq.isEmpty()) {
104               Vertice<E> vert = seq.dequeue();
105               Enlace<Arista<E>> enlacVert = vert.listArt.cabeza();
106               for(; enlacVert != null; enlacVert = enlacVert.siguiente) {
107                   if(enlacVert.valor.label == 0) {
108                       Vertice<E> w = enlacVert.valor.refDest;
109                       if(w.label == 0) {
110                           enlacVert.valor.label = 1;
111                           w.label = 1;
112                           sigSeq.enqueue(w);
113                       } else {
114                           enlacVert.valor.label = 2; //cross
115                       }
116                   }
117               }
118           }
119           seq = sigSeq;
120           i++;
121       }
122   }
```

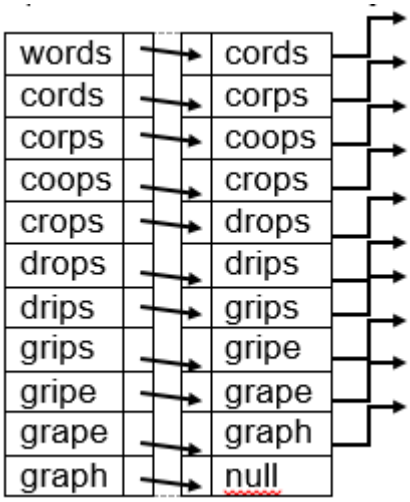
```java
private void DFSRecursivo(Vertice<E> v, int i) {
    v.label = 1;
    Enlace<Arista<E>> e = v.listArt.cabeza();
    for(; e != null; e = e.siguiente) {
        System.out.println("Analizando en "+v.data+" a "+e.valor.refDest.data);
        if(e.valor.label == 0) {
            Vertice<E> w = e.valor.refDest;
            if(w.label == 0) {
                e.valor.label = 1;
                DFSRecursivo(w, ++i);
            } else {
                e.valor.label = 2; //back
            }
        }
    }
}

```

```java
public void Dijkstra(E info) {
    PriorityQueue<Vertice<E>> q = new PriorityQueue<Vertice<E>>();

    Vertice<E> u = this.listVert.contiene(new Vertice<E>(info)).valor;

    Enlace<Vertice<E>> aux = this.listVert.cabeza();
    for(; aux!=null ; aux = aux.siguiente)  {
        if(aux.valor == u)
            aux.valor.dist = 0;
        else
            aux.valor.dist = 9999;
        aux.valor.path = null;
        aux.valor.label = 0;
        q.enqueue(aux.valor);
    }

    while(!q.isEmpty()) {
        System.out.println(q.toString());
        u = q.dequeue();
        u.label = 1;
        Enlace<Arista<E>> e = u.listArt.cabeza();
        for(;e != null; e = e.siguiente) {
            Vertice<E> z = e.valor.refDest;
            if(z.label == 0) {
                if(z.dist > (u.dist + e.valor.weight)) {
                    z.dist = u.dist + e.valor.weight;
                    z.path = u;
                }
            }
        }
        System.out.println(q.toString());
    }
    printDijkstra();
}
```

4. Solucionar el siguiente ejercicio: (5 puntos) El grafo de palabras se define de la siguiente manera: cada vértice es una palabra en el idioma Inglés y dos palabras son adyacentes si difieren exactamente en una posición. Por ejemplo, las cords y los corps son adyacentes, mientras que los corps y crops no lo son. a) Dibuje el grafo definido por las siguientes palabras: words cords corpscoops crops drops drips grips gripe grape graph



b) Mostrar la lista de adyacencia del grafo.



5. Realizar un método en la clase Grafo. Este método permitirá saber si un grafo está incluido en otro. Los parámetros de entrada son 2 grafos y la salida del método es true si hay inclusión y false el caso contrario. (4 puntos)

Para este método se hara un recorrido BFS en el grafo que queremos confirmar si esta incluido, donde se confirmara que estos mismos vértices y aristas estén presentes en el grafo original.

```
209             Vertice<E> vert = seq.dequeue();
210             Enlace<Arista<E>> enlacVert = vert.listArt.cabeza();
211             Vertice<E> vert2 = b.listVert.contiene(vert).valor;
212             if(vert2 == null)
213                 return false;
214             if(vert.equals(vert2)) {
215                 for(; enlacVert != null; enlacVert = enlacVert.siguiente) {
216 
217                     if((aux = vert2.listArt.contiene(enlacVert.valor)) != null) {
218                         if(aux.valor.weight == enlacVert.valor.weight) {
219                             if(enlacVert.valor.label == 0) {
220                                 Vertice<E> w = enlacVert.valor.refDest;
221                                 if(w.label == 0) {
222                                     enlacVert.valor.label = 1;
223                                     w.label = 1;
224                                     sigSeq.enqueue(w);
225                                 } else {
226                                     enlacVert.valor.label = 2; //cross
227                                 }
228                             }
229                         } else {
230                             System.out.println("Pesos Diferentes a1"+aux.valor.weight+"-- a2"+enlacVert.valor.weight);
231                             return false;
232                         }
233                     } else {
234                         System.out.println("Aristas diferentes ("+enlacVert.valor+")no existe en el grafo parametro");
235                         return false;
236                     }
237                 }
238             } else {
239                 System.out.println("Vertices Diferentes v1"+vert.data+"-- v2"+vert2.data);
240                 return false;
241             }
```

CUESTIONARIO

1. ¿Cuantas variantes del algoritmo de Dijkstra hay y cuál es la diferencia entre ellas? (1 puntos)

El método original de dijkstra encuentra la ruta más corta de un vértice a otro, mientras que hay una variante para de un solo nodo hallar la ruta más corta a todos los demas nodos, hay otra variante dirigida a los grafos dirigidos, donde se es cuidadoso al momento de actualizar etiquetas, tambien otra variante aplica a grafos no conexos donde si el peso es infinito se dice que no existe camino alguno o que se encuentra en otra componente.

2. Investigue sobre los ALGORITMOS DE CAMINOS MINIMOS e indique, ¿Qué similitudes encuentra, qué diferencias, en qué casos utilizar y porque? (2 puntos)

En clases de teoría se nos habló de los algoritmos: Dijkstra, Prim y Kruskal, Los dos primeros se parecen salvo que Dijkstra trabaja con el acumulado de pesos y Prim no hace eso. Kruskal por su parte va armando el árbol según una cola de prioridad de aristas, teniendo en cuenta de no romper las propiedades de los árboles. Al parecer Kruskal es el más rápido ya que solo se opera con las aristas y ya no se tendría que ir comparando nodo por nodo. Se recomiendo usar Prim para valores negativos, ya que dijkstra no produce una buena solución con negativos. Mientras que Prim y Kruskal solo funciona en grafos no dirigidos.