UNIVERSIDAD NACIONAL DE SAN AGUSTÍN
FACULTAD DE PRODUCCION Y SERVICIOS
ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

Curso: Laboratorio de Estructura de Datos

Estudiante:

-Ancasi Carlos,Boris Giovanni

Docente:

. Edith Pamela Rivero Tupac

2. Implementar el cogido de Grafo cuya representacion sea realizada mediante LISTA DE ADYACENCIA. (3 puntos)
Codigo hecho en el repositorio GITHUB por medio de lista enlazada
3. Implementar BSF, DFS y Dijkstra con sus respectivos casos de prueba. (5 puntos)
Codigo hecho en el repositorio GITHUB por medio de lista enlazada y colas
DFS por medio de recursividad

```
        private void DFSRec(Vertex<G> a) {
                a.label = 1;
                System.out.print(a.data+"   ;");
                Node<Edge<G>> enlace = a.listAdj.first;
                for(; enlace != null; enlace = enlace.getNext()) {
                        if(enlace.getData().label == 0) {
                                Vertex<G> vert = enlace.getData().refFinal;
                                if(vert.label == 0) {//no visitado
                                        enlace.getData().label = 1;//visitado
                                        DFSRec(vert);
                                }
                                else
                                        enlace.getData().label = 2;//tipo back
                        }
                }

        }
```

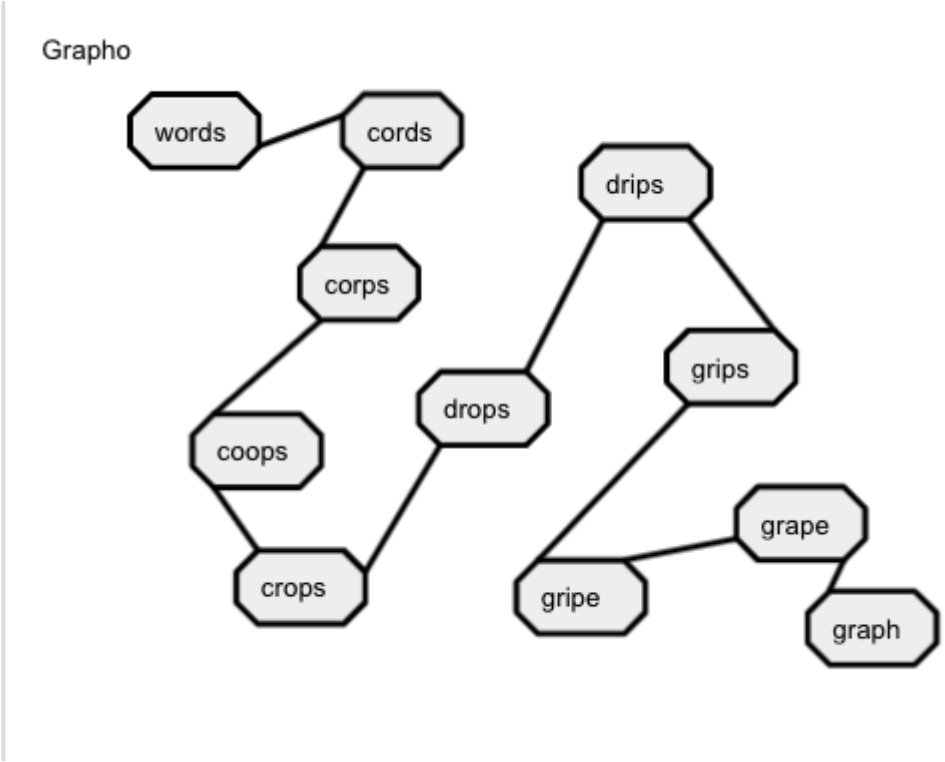
BFS por medio de colar de manera iterativa

```
        private void BFSRec(Vertex<G> a) {
                Cola<Vertex<G>> cola = new Cola<Vertex<G>>();
                cola.Encolar(a);
                a.label = 1;

                while(!cola.isEmpty()) {
                        Vertex<G> u = cola.desencolar();
                        System.out.print("papa: "+u.data+"\n");

                        Node<Edge<G>> enlace = u.listAdj.first;
                        System.out.print("hijos:   ");

                        for(; enlace != null; enlace = enlace.getNext()) {
                                if(enlace.getData().label == 0) {
                                        //cola.Encolar(enlace.getData().refFinal);
                                        Vertex<G> vert = enlace.getData().refFinal;
                                        if(vert.label == 0) {//no visitado
                                                System.out.print(vert.data+"   ");
                                                cola.Encolar(vert);
                                                enlace.getData().label = 1;//visitado
                                                vert.label = 1;
                                        }
                                        else
                                                enlace.getData().label = 2;//tipo back
```

```
                    }
                }
                System.out.println();
            }

        }
```

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 corps coops crops drops drips grips gripe grape graph



b) Mostrar la lista de adyacencia del grafo.

Lista de adyacencia

words ----->	**cords**
cords ----->	**words ------> corps**
corps ----->	**cords ------> coops**
coops ----->	**corps ------> crops**
crops ----->	**coops -------> drops**
drops ----->	**crops --------> drips**
drips ----->	**drops --------> grips**
grips ----->	**drips ------> gripe**
gripe ----->	**grips ------> grape**
grape ----->	**gripe ------> graph**
graph ----->	**grape**

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

Algoritmo Prim

Similar al algoritmo Dijkstra (para grafos conexos) que A partir de un vértice arbitrario s se obtiene el MST como una nube de vértices que Con cada vértice v se guarda una etiqueta d(v) igual al peso mínimo de las aristas que conectan v con un vértice en la nube
La única diferencia la señala la flecha, que es la función de relajación.
El Prim, que busca el árbol de expansión mínimo, solo se preocupa por el mínimo de los bordes totales que cubren todos los vértices.

Diferencias
La única diferencia que veo es que el algoritmo de Prim almacena una ventaja de costo mínimo, mientras que el algoritmo de Dijkstra almacena el costo total desde

un vértice de origen hasta el vértice actual.

Dijkstra le brinda un camino desde el nodo de origen al nodo de destino de manera que el costo sea mínimo. Sin embargo, el algoritmo de Prim le brinda un árbol de expansión mínimo, de modo que todos los nodos están conectados y el costo total es mínimo.

En palabras simples:

Entonces, si desea desplegar un tren para conectar varias ciudades, usaría el algoritmo de Prim. Pero si desea ir de una ciudad a otra ahorrando el mayor tiempo posible, usaría el algoritmo de Dijkstra.

Kruskal

El algoritmo de Prim se inicia con un nodo, mientras que el algoritmo de Kruskal se inicia con un borde.

• Los algoritmos de Prim abarcan desde un nodo a otro, mientras que el algoritmo de Kruskal selecciona los bordes de manera que la posición del borde no se base en el último paso.

• En el algoritmo de prim, el gráfico debe ser un gráfico conectado, mientras que el de Kruskal también puede funcionar en gráficos desconectados.

• El algoritmo de Prim tiene una complejidad de tiempo de O (V_2), y la complejidad del tiempo de Kruskal es O (logV).

Investigue sobre los ALGORITMOS DE CAMINOS MÍNIMOS e indique, ¿Qué similitudes encuentra, qué diferencias, en qué casos utilizar y porque?

algoritmo de floyd

También llamado algoritmo de caminos mínimos, es un algoritmo para la determinación del camino más corto dado un vértice origen al resto de los vértices en un grafo con pesos en cada arista

Encontrar el camino mas corto en una sola ejecución. Algoritmo que usa el

método de programación dinámica.

El procedimiento principal y único para este proceso recorre la matriz tantas veces como nodos tenga el grafo.

El algoritmo de Floyd puede ser utilizado para resolver los siguientes problemas:

- Camino mínimo en grafos dirigidos (algoritmo de Floyd).

- Aplicación de un medio de transporte , la cual encuentre la ruta optima o rápida para llegar a un destino especifico .

Algoritmo de Warshall

Encuentra si posible un camino entre cada uno de los vértices de la gráfica dirigida. Es decir no presenta las distancia entre los vértices Se basa en un concepto llamado cerradura transitiva de la matriz de adyacencia El algoritmo de Warshall sirve para encontrar la cerradura transitiva de una relación binaria en el conjunto A. La clausura transitiva de una relación binaria es la relación binaria mas pequeña que siendo transitiva contenga el conjunto de pares de la relación binaria original

El anterior algoritmo es poco eficiente, peor Cuando el grafo tiene muchos vértices Warshall propuso otro algoritmo mas eficiente

Algoritmo de Floyd-Warshall

El algoritmo de Floyd-Warshall compara todos los posibles caminos a través del grafo entre cada par de vértices. El algoritmo es capaz de hacer esto con sólo V^3 comparaciones (esto es notable considerando que puede haber hasta V^2 aristas en el grafo, y que cada combinación de aristas se prueba). Lo hace mejorando paulatinamente una estimación del camino más corto entre dos vértices, hasta que se sabe que la estimación es óptima.

Para que haya coherencia numérica, Floyd-Warshall supone que no hay ciclos negativos (de hecho, entre cualquier pareja de vértices que forme parte de un ciclo negativo, el camino mínimo no está bien definido porque el camino puede ser infinitamente pequeño). No obstante, si hay ciclos negativos,

Floyd-Warshall puede ser usado para detectarlos. Si ejecutamos el algoritmo una vez más, algunos caminos pueden decrementarse pero no garantiza que, entre todos los vértices, caminos entre los cuales puedan ser infinitamente pequeños, el camino se reduzca. Si los números de la diagonal de la matriz de caminos son negativos, es condición necesaria y suficiente para que este vértice pertenezca a un ciclo negativo.

Algoritmo de Bellman-Ford

El algoritmo de Bellman-Ford genera el camino más corto en un grafo dirigido ponderado (en el que el peso de alguna de las aristas puede ser negativo). El algoritmo de Dijkstra resuelve este mismo problema en un tiempo menor, pero requiere que los pesos de las aristas no sean negativos, salvo que el grafo sea dirigido y sin ciclos. Por lo que el Algoritmo Bellman-Ford normalmente se utiliza cuando hay aristas con peso negativo. Este algoritmo fue desarrollado por Richard Bellman, Samuel End y Lester Ford.

Las desventajas principales del algoritmo de Bellman-Ford en este ajuste son:

No escala bien

Los cambios en la topología de red no se reflejan rápidamente ya que las actualizaciones se distribuyen nodo por nodo.

Contando hasta el infinito(si un fallo de enlace o nodo hace que un nodo sea inalcanzable desde un conjunto de otros nodos, éstos pueden estar siempre aumentando gradualmente sus cálculos de distancia a él, y mientras tanto puede haber bucles de enrutamiento