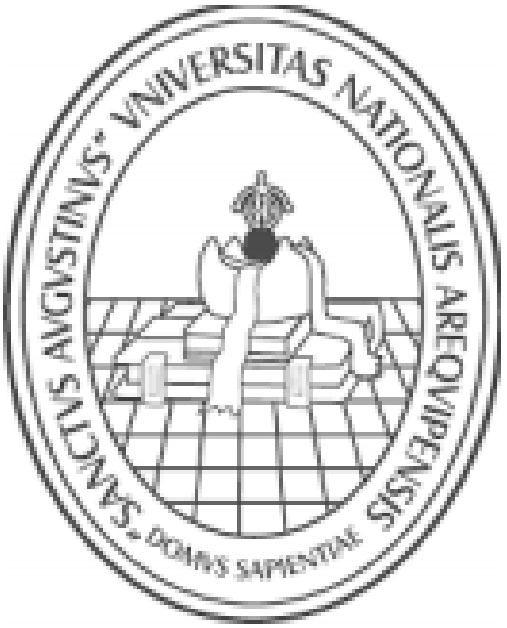
Universidad Nacional de San Agustín
Facultad de Ingeniería de Producción y Servicios
Escuela Profesional de Ingeniería de Sistemas



Lab Estructura de datos y algoritmos

LAB 9 GRAFOS

Docente : Ing. Edith Pamela Rivero Tupac

Nombre: Torres Aroquipa Karlo Jose

Arequipa - Perú
2021

1. Crear un repositorio en GitHub, donde incluyan la resolución de los ejercicios propuestos y el informe.

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

Clase Grafo

```java
package Grafos;
import java.util.*;

public class GrafoLAd {

    // funcion para añadir aristas, de donde inicio a fin, pero como no es dirigido
    //no importa mucho
    public static void addEdge(ArrayList<ArrayList<Integer> > adj, int u, int v){

        //como es no dirigido si a(nodo) es advacente a b, entonces b es advacente a
        adj.get(u).add(v);
        adj.get(v).add(u);
    }

    public static void printGraph(ArrayList<ArrayList<Integer> > adj)
    {
        for (int i = 0; i < adj.size(); i++) {
            System.out.println("lista de adyacencia del vertice: " + i);
            System.out.print( i);

            //a partir de aqui se imprimen los advacentes a la cabeza
            for (int j = 0; j < adj.get(i).size(); j++) {
                System.out.print(" -> "+adj.get(i).get(j));
            }
            System.out.println();
        }
    }


}
```

Clase Test

```java
import java.util.ArrayList;

public class Test {
        public static void main(String[] args){

            //cantidad de vertices
            int V = 4;
            //se crea un arraylist de los nodos advacentes al nodo examinado
            ArrayList<ArrayList<Integer> > adj= new ArrayList<ArrayList<Integer> >(V);

            //dependiendo de cada nodo yo añado un nuevo array list donde estaran sus advacentes
            for (int i = 0; i < V; i++)
                adj.add(new ArrayList<Integer>());

            GrafoLAd.addEdge(adj, 0, 1);
            GrafoLAd.addEdge(adj, 1, 3);
            GrafoLAd.addEdge(adj, 3, 2);
            GrafoLAd.addEdge(adj, 2, 0);


            GrafoLAd.printGraph(adj);
        }
}
```

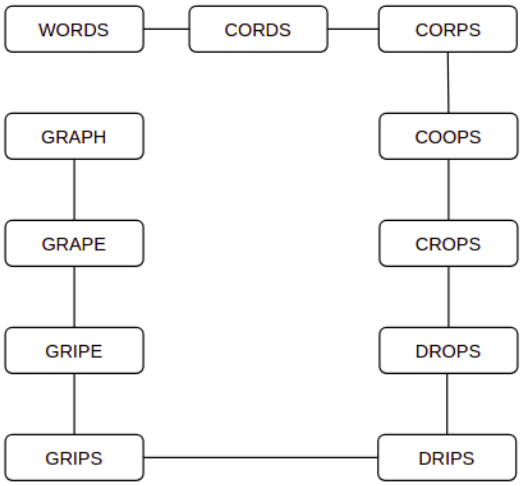
Salida(output)

```
<terminated> Test (4) [Java Applicatio
lista de adyacencia del vertice: 0
0 -> 1 -> 2
lista de adyacencia del vertice: 1
1 -> 0 -> 3
lista de adyacencia del vertice: 2
2 -> 3 -> 0
lista de adyacencia del vertice: 3
3 -> 1 -> 2
```

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

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

graph->grape

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 en caso contrario. (4 puntos)

CUESTIONARIO

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

Dijkstra con cola de prioridad

Con esta implementación utiliza O(m log n) operaciones, siendo n la cantidad de nodos del problema y m la cantidad de aristas, se basa en que las estructuras que usamos para la cola de prioridad extraen e insertan un elemento en tiempo logarítmico.
Si el grafo tiene pocas aristas, conviene utilizar la implementación con cola de prioridad (O(m log n))
Si el grafo tiene muchas aristas, conviene utilizar la implementación básica (O(n^2))

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

a. **Algoritmo de Dijkstra:**
   También llamado algoritmo de caminos mínimos, es un algoritmo para la determinación del camino más corto dado un vértice origen al resto de vértices en un grafo con pesos en cada arista. Su nombre se refiere a Edsger Dijkstra, quien lo describió por primera vez en 1959. Tiene una complejidad O(v^2)
   Aplicaciones:
   - Distribución de productos a una red de establecimientos comerciales.
   - Distribución de correos postales.

b. **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 Bellman-Ford normalmente se utiliza cuando hay aristas con peso negativo. Este algoritmo fue desarrollado por Richard Bellman, Samuel End y Lester Ford. Tiene una complejidad de O(VE) donde V: vértices y E: aristas.

   El algoritmo Dijkstra resuelve este mismo problema en un tiempo menor, pero que los pesos de las aristas sean positivos.
   Este algoritmo puede ser utilizado para realizar un mapa de todas las posibles rutas que se pueden tomar desde un punto a los demás; esto es aplicable a todos los dispositivos de ruteo locales que se utilizan en la red.

c. **Algoritmo de floyd Warshall:**

El algoritmo de Floyd-Warshall, descrito en 1959 por Bernard Roy, es un algoritmo de análisis sobre grafos para encontrar el camino mínimo en grafos dirigidos ponderados. El algoritmo encuentra el camino entre todos los pares de vértices en una única ejecución. El algoritmo de Floyd-Warshall es un ejemplo de programación dinámica.
La complejidad de este algoritmo es 0(n^3). El algoritmo resuelve eficientemente la búsqueda de todos los caminos más cortos entre cualesquiera nodos.
La diferencia con los anteriores es que que Floyd Warshall puede trabajar con pesos positivos y negativos