Guía de Actividades Práctico-Experimentales Nro. 010

1. Datos Generales

Asignatura	Estructura de datos
Ciclo	3 A
Unidad	3
Resultado de aprendizaje de la unidad	Entiende los principios básicos de las operaciones en árboles, bajo los principios de solidaridad, transparencia, responsabilidad y honestidad.
Título de la Práctica	Implementación algoritmo de Dijkstra en grafos ponderados
Nombre del Docente	Andrés Roberto Navas Castellanos
Fecha	Jueves 29 de enero
Horario	07h30 – 10h30
Lugar	Aula
Tiempo planificado en el Sílabo	3 horas

2. Objetivo(s) de la Práctica:

- Comprender el problema del camino más corto en grafos ponderados.
- Implementar correctamente el algoritmo de Dijkstra en Java.
- Interpretar las estructuras auxiliares del algoritmo (distancias, visitados, predecesores).
- Verificar la ejecución manual frente a la salida del programa.

3. Materiales y reactivos:

- Archivo de entrada con un grafo ponderado (matriz o lista).

4. Equipos y herramientas

- JDK OpenJDK (obligatorio).
- IDE: Visual Studio Code (extensión "Extension Pack for Java") o IntelliJ IDEA Community.
- Sistema de control de versiones: Git; repositorio en GitHub.
- EVA/Moodle institucional: para entrega de evidencias.
- Herramientas de documentación: README Markdown, editor ofimático (Google Docs/LibreOffice/Word).

5. Procedimiento / Metodología

Enfoque metodológico: ABPr (Aprendizaje Basado en Proyectos).

Inicio

- Presentación del problema real: encontrar la ruta más corta entre dos puntos (ej. red vial, red de servicios, sistema de atención).
- Análisis del grafo proporcionado: nodos, aristas y pesos.
- Recordatorio conceptual del algoritmo de Dijkstra (sin entrar aún en código).

Desarrollo

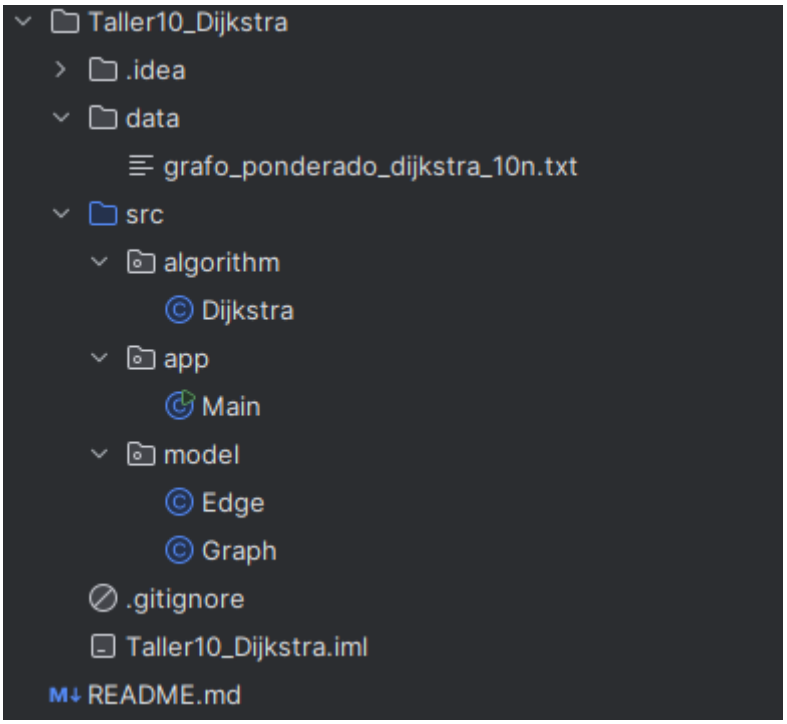
- Representar el grafo ponderado en Java.
- Definir las estructuras necesarias:
  - Arreglo de distancias mínimas.
  - Arreglo de nodos visitados.
  - Arreglo de predecesores.
- Implementar el algoritmo de Dijkstra desde un nodo origen.
- Ejecutar el algoritmo paso a paso:
  - Selección del nodo con menor distancia no visitado.
  - Relajación de aristas.
- Mostrar en consola:
  - Distancias finales.
  - Camino mínimo reconstruido.
- Comparar la salida del programa con la ejecución manual realizada en clase.

Cierre

- Análisis conjunto de los resultados obtenidos.
- Discusión entre miembros del grupo
  - Por qué el algoritmo funciona.
  - Qué ocurre si hay pesos negativos.
- Retroalimentación del docente sobre la implementación y lógica utilizada.

6. Resultados esperados:

- Código Java funcional del algoritmo de Dijkstra.
- Salida en consola con distancias y caminos.




```java
package app;

import ...

public class Main {  Ismael

    public static void main(String[] args) {  Ismael

        Scanner input = new Scanner(System.in);
        File dataFolder = new File( pathname: "data");

        if (!dataFolder.exists() || !dataFolder.isDirectory()) {
            System.out.println("La carpeta 'data' no existe.");
            return;
        }

        boolean running = true;

        while (running) {

            File[] files = dataFolder.listFiles(( File dir,  String name) -> name.endsWith(".t

            if (files == null || files.length == 0) {
                System.out.println("No hay archivos .txt en la carpeta data.");
                return;
            }

            // ===== MENÚ DE ARCHIVOS =====
            System.out.println("\nArchivos disponibles:");
            for (int i = 0; i < files.length; i++) {
```

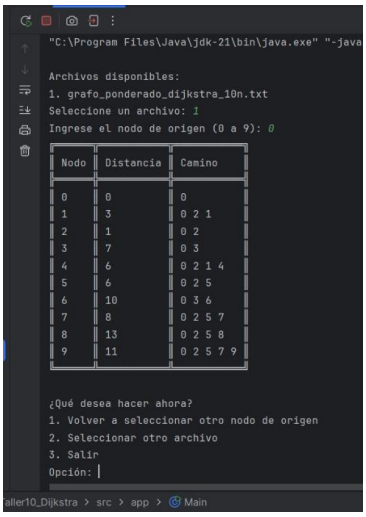
```java
package algorithm;

import model.Edge;
import model.Graph;

import java.util.Arrays;

public class Dijkstra {  2 usages  Ismael

    private static final int INF = Integer.MAX_VALUE;

    public static void run(Graph graph, int source) {

        int n = graph.getN();
        int[] dist = new int[n];
        boolean[] visited = new boolean[n];
        int[] prev = new int[n];

        Arrays.fill(dist, INF);
        Arrays.fill(prev,  val: -1);
        dist[source] = 0;

        for (int i = 0; i < n - 1; i++) {
            int u = minDistance(dist, visited);
            if (u == -1) break;

            visited[u] = true;

            for (Edge e : graph.getAdj().get(u)) {
```

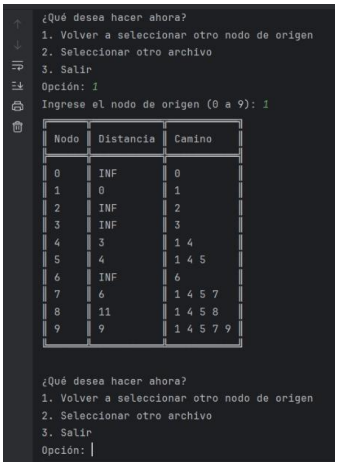
```java
package model;

public class Edge {  5 usages  Ismael
    public int to;  2 usages
    public int weight;  2 usages

    public Edge(int to, int weight) {
        this.to = to;
        this.weight = weight;
    }
}
```

```java
public class Graph {  5 usages  Ismael

    private int n;  3 usages
    private List<List<Edge>> adj;  4 usages

    public Graph(String filePath) throws FileNotFoundException {

        Scanner sc = new Scanner(new File(filePath));

        n = sc.nextInt();
        int m = sc.nextInt();

        adj = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            adj.add(new ArrayList<>());
        }

        for (int i = 0; i < m; i++) {
            int u = sc.nextInt();
            int v = sc.nextInt();
            int w = sc.nextInt();
            adj.get(u).add(new Edge(v, w));
        }

        sc.close();
    }
```

```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2025.2.4\lib\idea_rt.jar=56946" -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF

Archivos disponibles:
1. grafo_ponderado_dijkstra_10n.txt
Seleccione un archivo:
```

```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2025.2.4\lib\idea_rt.jar=56946" -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF

Archivos disponibles:
1. grafo_ponderado_dijkstra_10n.txt
Seleccione un archivo: 1
Ingrese el nodo de origen (0 a 9):
```





Link Del Git: https://github.com/IsmaelGonz/Taller10-Implementacion_algoritmo_de_Dijkstra_en_grafos_ponderados

7. Preguntas de Control:

- **¿Cuál es el objetivo principal del algoritmo de Dijkstra?**

  Encontrar el camino más corto desde un nodo origen hacia todos los demás nodos en un grafo ponderado con pesos positivos
- **¿Qué significa "relajar" una arista?**

  Comparar si pasar por una arista mejora la distancia actual hacia un nodo destino y, si es así, actualizar la distancia y su predecesor.
- **¿Por qué Dijkstra no funciona con pesos negativos?**

  Porque el algoritmo asume que, una vez fijada la menor distancia a un nodo, esa distancia no puede disminuir, lo cual no se cumple si existen pesos negativos.
- **¿Qué estructura permite reconstruir el camino mínimo?**

  El arreglo de predecesores, que guarda el nodo anterior en el camino más corto hacia cada nodo.
- **¿En qué tipo de problemas reales se aplica este algoritmo?**

  Se usa en problemas como, rutas en mapas y GPS, redes viales, redes de comunicación, planificación de rutas en sistemas de transporte y servicios

8. Evaluación

Criterio	Excelente	Bueno	Básico	Puntaje
Implementación del algoritmo	Correcta y completa	Error menor	Parcial	4
Ejecución y validación	Correcta y justificada	Parcial	Débil	3
Análisis de resultados	Claro y bien argumentado	Aceptable	Superficial	2
Claridad del código	Muy clara	Aceptable	Desordenada	1

9. Bibliografía

[1] T. H. Cormen et al., Introduction to Algorithms, 4th ed., MIT Press, 2022.

[2] M. A. Weiss, Data Structures and Algorithm Analysis in Java, 4th ed., Pearson, 2021.

[3] E. W. Dijkstra, "A note on two problems in connexion with graphs," Numerische Mathematik, 1959.

10. Elaboración y Aprobación

Elaborado por	Andrés R Navas Castellanos **Docente**	Firmado electrónicamente por: ANDRES ROBERTO NAVAS CASTELLANOS Validar únicamente con FirmaEC
Revisado por **Solo si es realizado en laboratorios**	Luis Sinche **Técnico Docente**	No Aplica
Aprobado por	Edison L Coronel Romero **Director de Carrera**	Firmado electrónicamente por: EDISON LEONARDO CORONEL ROMERO Validar únicamente con FirmaEC