

**UNIVERSIDAD DE SONORA**

**Unidad Regional Centro**

**División de Ingeniería**

# **Ingeniería en Sistemas de Información**

**Fundamentos de Computación 3**

**Maestro: IVAN DOSTOYEWSKI MEZA IBARRA**

**Grupo 1 – Hora 5 PM – 6 PM- Edif. 5G – L205**

**Nombre del Alumno:**

* **Vallejo Leyva Marcos**

**Act.7 Implementar algoritmo de búsqueda de ruta corta en grafo.**

# **Act.7 Implementar algoritmo de búsqueda de ruta corta en grafo.**

**Índice**

1. **Introducción**
2. **Marco Teórico**
3. **Descripción General del Código**
4. **Métodos Implementados en el Código**
5. **Ejecución y código**
6. **Conclusiones**

**Introducción**

Este documento describe un programa en Java diseñado para encontrar la ruta más corta en un grafo utilizando dos algoritmos conocidos: Dijkstra y Bellman-Ford. Estos algoritmos son esenciales en el campo de las ciencias de la computación, especialmente en el diseño de redes y sistemas de navegación. El programa permite al usuario crear un grafo, añadir aristas con pesos y seleccionar un algoritmo para calcular la distancia más corta desde un vértice de origen a todos los demás vértices del grafo.

**Marco Teórico**

**Grafo:** Un grafo es una estructura matemática que consiste en un conjunto de vértices (también llamados nodos) y un conjunto de aristas (también llamadas bordes) que conectan estos vértices. Los grafos se utilizan ampliamente en ciencias de la computación y en diferentes áreas para modelar una variedad de situaciones y relaciones entre objetos.

* **Algoritmo de Dijkstra:** El algoritmo de Dijkstra es utilizado para encontrar el camino más corto desde un nodo a todos los otros nodos en un grafo ponderado sin ciclos de peso negativo. Utiliza una cola de prioridad para explorar el nodo con la distancia más corta acumulada primero.
* **Algoritmo de Bellman-Ford:** El algoritmo de Bellman-Ford también encuentra el camino más corto en grafos ponderados y puede manejar grafos con ciclos de peso negativo, reportando si existen estos ciclos. Funciona mediante la relajación repetida de todas las aristas.

**Descripción General del Código**

El código consiste en dos clases principales: **Graph** y **Edge**, y una clase **Main** que maneja la entrada del usuario y la ejecución de los algoritmos.

* **Clase Graph:** Gestiona la creación del grafo y la ejecución de los algoritmos de Dijkstra y Bellman-Ford.
* **Clase Edge:** Representa las aristas del grafo, almacenando los vértices de origen y destino, y el peso de la arista.

**Métodos Implementados en el Código**

**Clase Graph**

La clase **Graph** incluye una variedad de métodos esenciales para gestionar un grafo y ejecutar algoritmos de ruta más corta:

Texto

Descripción generada automáticamente

* **addEdge(int source, int dest, int weight)**: Este método añade una arista al grafo dirigido desde un vértice **source** a un vértice **dest** con un peso específico **weight**. La arista se representa como un objeto de la clase **Edge** y se añade a la lista de adyacencia del vértice origen.

Texto

Descripción generada automáticamente

* **dijkstra(int startVertex)**: Implementa el algoritmo de Dijkstra utilizando una cola de prioridad para seleccionar siempre el vértice con la distancia estimada más corta aún no visitada. Este método inicializa las distancias de todos los vértices a infinito, excepto el vértice inicial que se establece en cero. A medida que se exploran los vértices, se actualizan las distancias mínimas acumuladas y se almacenan en un arreglo de distancias.

Texto

Descripción generada automáticamente

* **bellmanFord(int startVertex)**: Ejecuta el algoritmo de Bellman-Ford, que es útil especialmente para grafos que contienen ciclos de peso negativo. Este método trabaja relajando repetidamente todas las aristas y actualizando las distancias mínimas para cada vértice. Además, realiza una verificación adicional después de todas las relajaciones para detectar ciclos de peso negativo en el grafo.

Texto

Descripción generada automáticamente

* **printSolution(int[] distances, int startVertex)**: Este método imprime las distancias del vértice inicial a todos los demás vértices. Si alguna distancia permanece como infinito, se imprime como "Infinity", indicando que el vértice no es alcanzable desde el vértice de inicio.

Texto

Descripción generada automáticamente

**Clase Edge**

* **Constructor Edge(int source, int dest, int weight)**: Este constructor inicializa un objeto **Edge** con un vértice de origen (**source**), un vértice de destino (**dest**), y un peso (**weight**). Este diseño facilita la gestión de aristas dirigidas y ponderadas dentro del grafo.

Texto

Descripción generada automáticamente

**Ejecución y Código:**

La clase **Main** gestiona la interacción con el usuario y controla la ejecución del programa. La ejecución del programa se desarrolla de la siguiente manera:

1. **Inicialización del Grafo**: El programa comienza solicitando al usuario el número de vértices y, en base a esto, inicializa un objeto **Graph**.
2. **Ingreso de Aristas**: Posteriormente, se pide al usuario que especifique el número de aristas y se le solicita que introduzca cada una de ellas en el formato **<source> <destination> <weight>**. Para cada arista ingresada, se llama al método **addEdge** de la clase **Graph** para agregar la arista al grafo.
3. **Selección del Vértice Inicial y el Algoritmo**: El usuario debe especificar el vértice inicial desde donde se buscarán las rutas más cortas. Luego, el usuario elige entre ejecutar el algoritmo de Dijkstra o Bellman-Ford.
4. **Ejecución del Algoritmo**:
   * Si se selecciona Dijkstra, el método **dijkstra** de la clase **Graph** se ejecuta para calcular las distancias más cortas desde el vértice inicial a todos los otros vértices usando este algoritmo.
   * Si se selecciona Bellman-Ford, se ejecuta el método **bellmanFord** para calcular las distancias y detectar posibles ciclos de peso negativo.
5. **Impresión de Resultados**: Después de ejecutar el algoritmo seleccionado, el método **printSolution** se llama para mostrar las distancias desde el vértice inicial a todos los otros vértices, permitiendo al usuario visualizar los resultados de la ruta más corta.
6. **Manejo de Entradas Inválidas**: El programa también incluye control de entradas inválidas, como la selección de un algoritmo no válido, asegurando que el usuario reciba retroalimentación adecuada para corregir errores de entrada.

**Código clase Main:**

Texto

Descripción generada automáticamente

**Ejecución del código:**

Texto

Descripción generada automáticamente Texto

Descripción generada automáticamente

**Conclusiones:**

Este programa muestra una herramienta efectiva para la visualización y comprensión de dos algoritmos críticos de ruta más corta en la teoría de grafos. Permite a los usuarios entender mejor cómo estos algoritmos abordan el problema de la ruta más corta bajo diferentes condiciones y restricciones del grafo. Esta aplicación es especialmente útil en el ámbito educativo y para aquellos que buscan profundizar en el análisis de algoritmos y estructuras de datos.