Programación III

Trabajo Práctico Especial

Curso: 424608	Aula: P107	Turno: MRI
Integrantes:		
1.	Gil, Gonzalo Damian	LU: 1114910
2.	Shilman, Walter Elias	LU: 1107840
Profesor: Rodríguez, Guillermo Horacio		
Fecha: 02/03/23	Cuatrimestre: 1er 2023	

Se implementarán en lenguaje Java los siguientes 3 algoritmos:

- Algoritmo Depth-First Search (DFS).
- Algoritmo de Floyd.
- Algoritmo Dijkstra.

Algoritmo Depth-First Search (DFS)

Realiza la búsqueda en profundidad en un grafo y devuelve el árbol de búsqueda resultante en forma de un arreglo de padres.

```
public static int[] DFS(ImplemEstatica grafo, int origen, int[] p, String

    // Marcando el v?rtice como descubierto
    marca[origen] = "G";

    // Recorriendo los adyacentes al v?rtice origen
    for (int v : grafo.adyacentes(origen))
        if (marca[v] == "B") {
            // Agregando el padre al arreglo de padres
            p[v] = origen;
            // Llamando recursivamente a DFS
            DFS(grafo, v, p, marca);
        }

    // Cambiando la marca a visitado
    marca[origen] = "N";

    // Retornando el arreglo de padres
    return p;
}
```

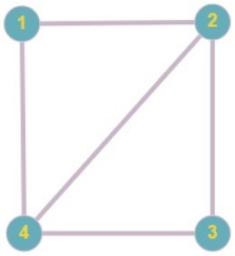
Código

Éste método implementa el algoritmo de búsqueda en profundidad (DFS) en un grafo representado por una estructura de datos de tipo ImplemEstatica, que suponemos contiene una lista de adyacencia de los vértices del grafo.

La función toma como entrada: el grafo, un vértice origen desde el cual comenzar la búsqueda, un arreglo de padres (p) que se actualizará durante la búsqueda para reflejar el árbol de búsqueda resultante, y un arreglo de marcas de vértices (marca) que indica si un vértice ha sido descubierto (G), visitado (N) o no visitado (B).

Comienza marcando el vértice origen como descubierto (G). Luego, para cada vértice adyacente al origen que no haya sido descubierto, se agrega el origen como padre y se llama recursivamente a DFS desde el vértice adyacente. Esto se hace hasta que no haya más vértices adyacentes por descubrir.
Una vez que se han explorado todos los vértices adyacentes al origen, se marca el vértice origen como visitado (N) y se devuelve el arreglo de padres p actualizado. El árbol de búsqueda resultante se puede construir a partir de este arreglo de padres.

Se utilizó este grafo para probar el algoritmo:



Algoritmo de Floyd

```java
public static int[][] calcularFloyd(int matriz[][]) {

    for (int i = 0; i < matriz.length; i++) {
        for (int j = 0; j < matriz.length; j++) {
            if(matriz[i][j] == 0) {
                matriz[i][j] = 99999;
            }
            if(i == j) {
                matriz[i][j] = 0;
            }
        }
    }

    for (int i = 0; i < matriz.length; i++) {
        for (int j = 0; j < matriz.length; j++) {
            for (int k = 0; k < matriz.length; k++) {
                if(matriz[j][i] + matriz[i][k] < matriz[j][k]) {
                    matriz[j][k] = matriz[j][i] + matriz[i][k];
                }
            }
        }
    }

    return matriz;
}
```

Código

Este código implementa el algoritmo de Floyd-Warshall para encontrar la distancia más corta entre todos los pares de nodos en un grafo ponderado no dirigido representado por una matriz de adyacencia.

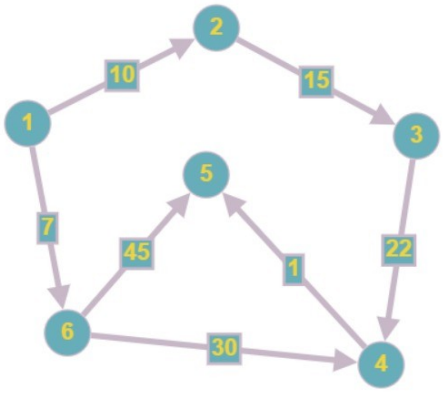
La función toma como entrada una matriz de adyacencia que representa el grafo ponderado y devuelve una matriz que contiene las distancias más cortas entre todos los pares de nodos.

El algoritmo comienza inicializando los valores de la diagonal principal de la matriz de adyacencia como 0 y los valores restantes como infinito (en este caso, 99999). Esto es porque la distancia de un nodo a sí mismo es siempre 0, y cualquier distancia desconocida se asume como infinito.

Luego, se aplica el algoritmo de Floyd en dos bucles anidados para actualizar las distancias más cortas entre todos los pares de nodos. En cada iteración, se considera un nodo intermedio entre dos nodos y se actualiza la distancia más corta entre ellos si es más corta que la distancia actual.

Este proceso se repite para todos los nodos intermedios posibles y se actualizan las distancias correspondientes en la matriz de salida. Al finalizar el algoritmo, se devuelve la matriz de salida que contiene las distancias más cortas entre todos los pares de nodos.

Se utilizó este grafo para probar el algoritmo:



Algoritmo de Dijkstra

```java
public int[] dijkstra(int origen) {
    int n = this.dim;
    int[] distancia = new int[n];
    boolean[] visitados = new boolean[n];
    int[] ruta = new int[n];

    // Inicializar los valores de distancia y ruta
    for (int i = 0; i < n; i++) {
        distancia[i] = Integer.MAX_VALUE;
        ruta[i] = -1;
    }
    distancia[this.posicionDeNodo(origen)] = 0;

    // Iterar hasta que se hayan visitado todos los nodos
    for (int i = 0; i < n; i++) {
        // Encontrar el nodo no visitado con la distancia m?nima
        int minDistancia = Integer.MAX_VALUE;
        int minNodo = -1;
        for (int j = 0; j < n; j++) {
            if (!visitados[j] && distancia[j] < minDistancia) {
                minDistancia = distancia[j];
                minNodo = j;
            }
        }
        if (minNodo == -1) {
            break;
        }

        // Marcar el nodo como visitado
        visitados[minNodo] = true;

        // Actualizar las distancias a los nodos vecinos
        for (int j = 0; j < n; j++) {
            int peso = matrizAdy[minNodo][j];
            if (peso > 0 && !visitados[j]) {
                int nuevaDistancia = distancia[minNodo] + peso;
                if (nuevaDistancia < distancia[j]) {
                    distancia[j] = nuevaDistancia;
                    ruta[j] = minNodo;
                }
            }
        }
    }

    // Retornar las distancias desde el origen y la ruta
    return distancia;
}
```

Código

Éste método implementa el algoritmo de Dijkstra para encontrar la ruta más corta desde un nodo origen hasta todos los demás nodos en un grafo ponderado no dirigido representado por una matriz de adyacencia.

La función toma como entrada el nodo origen desde el cual se desea encontrar la ruta más corta. La función devuelve un arreglo de distancias desde el origen hasta todos los demás nodos y un arreglo de rutas que indica el camino más corto desde el origen hasta cada nodo.

La función comienza inicializando las distancias de todos los nodos como "infinitas" excepto para el nodo origen, cuya distancia se establece en cero. También se inicializan los arreglos de visitados y rutas.

Luego, se itera a través de todos los nodos no visitados hasta que se hayan visitado todos los nodos. En cada iteración, se encuentra el nodo no visitado con la distancia mínima desde el origen y se marca como visitado.

Se actualizan las distancias a todos los nodos vecinos que no han sido visitados, y se actualiza la ruta si se encuentra una ruta más corta. Este proceso se repite hasta que se hayan visitado todos los nodos o no se encuentre un nodo no visitado con una distancia finita.

Finalmente, la función devuelve el arreglo de distancias y el arreglo de rutas. Los valores en el arreglo de distancias indican la distancia más corta desde el origen hasta cada nodo, y los valores en el arreglo de rutas indican el camino más corto desde el origen hasta cada nodo en términos de los nodos visitados en el camino.

Se utilizó este grafo para probar el algoritmo: