Programación III

Trabajo Práctico Especial

Curso: 424608	Aula: P107	Turno: MRI
Integrantes:		
1.	Ansede, Guido Nicolas	LU: 1114075
2.	Piersanti, Lucas Gabriel	LU: 110514
3	Valenzuela Arari, Marcos David	LU: 1114907
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 Prim.

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;
}
```

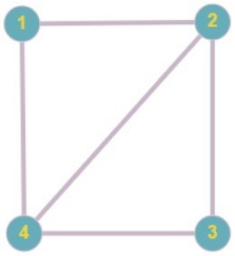
Código

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.

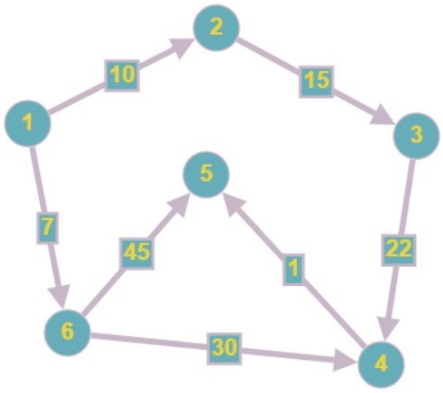
La función toma como entrada una matriz de adyacencia que representa el grafo ponderado y devuelve una matriz que contiene las distancias más cortas entre todos los pares de nodos.

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 Prim

```java
public ImplemEstatica prim(int verticeInicial) {
    ImplemEstatica agm = new ImplemEstatica();
    agm.inicializarGrafo(dim);
    agm.agregarVertice(etiquetas[verticeInicial]);
    while (agm.indice < dim) {
        int pesoMinimo = Integer.MAX_VALUE;
        int verticeAgregado = -1;
        int verticeConectado = -1;
        for (int i = 0; i < agm.indice; i++) {
            for (int j = 0; j < dim; j++) {
                if (matrizAdy[i][j] != 0 && !agm.pertenece(etiquetas[j])) {
                    if (matrizAdy[i][j] < pesoMinimo) {
                        pesoMinimo = matrizAdy[i][j];
                        verticeAgregado = j;
                        verticeConectado = i;
                    }
                }
            }
        }

        if (verticeConectado >= 0 && verticeAgregado >= 0) {
            agm.agregarVertice(etiquetas[verticeAgregado]);
            agm.agregarArista(etiquetas[verticeConectado], etiquetas[verticeAgregado], pesoMinimo);
        }

        agm.agregarArista(etiquetas[verticeConectado], etiquetas[verticeAgregado], pesoMinimo);
    }
    return agm;
}
```

Código

Este método implementa el algoritmo de Prim para obtener el árbol de recubrimiento minimo de un grafo (armc) no dirigido y conexo representado por una matriz de adyacencia.

El algoritmo comienza creando un nuevo objeto ImplemEstatica (grafo estático) para almacenar el armc resultante. Luego se inicializa el grafo con el tamaño del grafo original representado por la variable dim y se agrega el primer vértice indicado por el parámetro verticeInicial a la lista de vértices del armc.

A continuación, se entra en un bucle while que se repetirá hasta que todos los vértices del grafo original hayan sido agregados al armc. En cada iteración del bucle, se busca el vértice no incluido en el armc que tenga la menor arista que lo conecte a un vértice que sí esté en el armc. Para ello se recorre la matriz de adyacencia, y se compara cada arista con el peso mínimo actual (pesoMinimo) y se actualiza este valor y los vértices involucrados si se encuentra una arista de menor peso.

Si se encuentra una arista que cumpla las condiciones anteriores, se agrega el vértice no incluido en el armc (verticeAgregado) a la lista de vértices del armc y se agrega la arista mínima que lo conecta al

vértice del AGM (verticeConectado), usando los métodos agregarVertice y agregarArista respectivamente.

Finalmente, el método devuelve el objeto ImplemEstatica que contiene el armc resultante.

Se utilizó este grafo para probar el algoritmo: