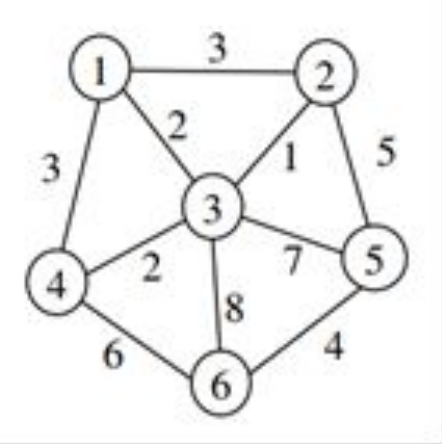
Algoritmo de Kruskal para Árbol de Expansión Mínima

Conectando puntos al menor costo

¿Qué es Kruskal?

Un algoritmo de teoría de grafos del tipo Greedy, se utiliza para encontrar el Árbol de Expansión Mínima (MST) en un grafo. El objetivo es conectar todos los nodos del grafo usando las aristas de menor peso posible sin formar ciclos. Esto se logra seleccionando las aristas de forma ordenada según su peso, asegurándose de que cada nueva conexión no cierre un ciclo.



Enunciado

Aplicar el algoritmo de Kruskal sobre el siguiente grafo, mostrando el orden en que son añadidas las aristas a la solución.

Implementación en JAVA

Nodo

```java
package Estructura;

import java.util.ArrayList;
import java.util.List;

6 usages  Dipacce Bono
public class Nodo {
    3 usages
    private int id;
    4 usages
    private List<Arista> adyacencias;

    1 usage  Dipacce Bono
    public Nodo(int id) {
        this.id = id;
        this.adyacencias = new ArrayList<>();
    }

    2 usages  Dipacce Bono
    public int getId() {
        return id;
    }

    2 usages  Dipacce Bono
    public List<Arista> getAdyacencias() {
        return adyacencias;
    }

    2 usages  Dipacce Bono
    public void agregarArista(Arista arista) {
        adyacencias.add(arista);
    }

    Dipacce Bono
    @Override
    public String toString() {
        return "Nodo " + id + " conectado a: " + adyacencias;
    }
}
```

Arista

```java
package Estructura;

20 usages  Dipacce Bono
public class Arista {
    3 usages
    private int origen;
    3 usages
    private int destino;
    3 usages
    private int peso;

    2 usages  Dipacce Bono
    public Arista(int origen, int destino, int peso) {
        this.origen = origen;
        this.destino = destino;
        this.peso = peso;
    }

    4 usages  Dipacce Bono
    public int getOrigen() { return origen; }

    5 usages  Dipacce Bono
    public int getDestino() { return destino; }

    5 usages  Dipacce Bono
    public int getPeso() { return peso; }

    Dipacce Bono
    @Override
    public String toString() { return "(" + origen + " -> " + destino + ", peso: " + peso + ")"; }
}
```

Grafo Dinámico

Implementación en JAVA

```java
package Estructura;

import java.util.*;

4 usages  Dipacce Bono
public class GrafoDinamico {
    8 usages
    private Map<Integer, Nodo> nodos;

    1 usage  Dipacce Bono
    public GrafoDinamico() { nodos = new HashMap<>(); }

    2 usages  Dipacce Bono
    public void agregarNodo(int id) { nodos.putIfAbsent(id, new Nodo(id)); }

    10 usages  Dipacce Bono
    public void agregarArista(int origen, int destino, int peso) {
        agregarNodo(origen);
        agregarNodo(destino);

        Arista arista = new Arista(origen, destino, peso);
        nodos.get(origen).agregarArista(arista);

        Arista aristaInversa = new Arista(destino, origen, peso);
        nodos.get(destino).agregarArista(aristaInversa);
    }

    no usages  Dipacce Bono
    public void mostrarGrafo() {
        for (Nodo nodo : nodos.values()) {
            System.out.println(nodo);
        }
    }

    4 usages  Dipacce Bono
    public List<Arista> getAristas() {
        Set<Arista> aristasUnicas = new HashSet<>();
        for (Nodo nodo : nodos.values()) {
            aristasUnicas.addAll(nodo.getAdyacencias());
        }
        return new ArrayList<>(aristasUnicas);
    }
```

Representa un grafo no dirigido y dinámico, permitiendo agregar nodos, aristas y mostrar la estructura del grafo. Es adecuada para gestionar grafos donde las conexiones cambian o se agregan de forma flexible.

- `agregarNodo(int id)`: Agrega un nodo con el identificador dado al grafo, si aún no existe.
- `agregarArista(int origen, int destino, int peso)`: Crea y agrega una arista entre dos nodos con un peso específico, agregando los nodos si aún no existen en el grafo.
- `mostrarGrafo()`: Imprime en consola los nodos y sus conexiones, mostrando la estructura del grafo.
- `getAristas()`: Devuelve una lista de todas las aristas del grafo, eliminando duplicados en caso de grafos no dirigidos.
- `getNodos()`: Devuelve una colección de todos los nodos del grafo.

Implementación en JAVA

```java
public List<Arista> kruskalMST() {
    List<Arista> mst = new ArrayList<>();
    UnionFind uf = new UnionFind();

    for (Integer nodoId : nodos.keySet()) {
        uf.makeSet(nodoId);
    }


    List<Arista> aristas = getAristas();
    Collections.sort(aristas, Comparator.comparingInt(Arista::getPeso));


    for (Arista arista : aristas) {
        int origen = arista.getOrigen();
        int destino = arista.getDestino();

        if (uf.find(origen) != uf.find(destino)) {
            mst.add(arista);
            uf.union(origen, destino);
        }
    }
    return mst;
}
```

Este método encuentra el Árbol de Expansión Mínima (MST) usando el algoritmo de Kruskal en tres pasos:

1. **Inicialización**:
   - Crea un conjunto independiente para cada nodo con la estructura `UnionFind`, permitiendo verificar conexiones sin ciclos.
2. **Ordenación de Aristas**:
   - Ordena todas las aristas por peso, de menor a mayor, para priorizar las conexiones más baratas.
3. **Construcción del MST**:
   - Recorre las aristas ordenadas y agrega al MST aquellas que conectan nodos de conjuntos diferentes, evitando ciclos.

Al final devuelve el MST, que conecta todos los nodos con el peso total mínimo.

Implementación en JAVA

```java
public UnionFind() {
    parent = new HashMap<>();
    rank = new HashMap<>();
}

1 usage  Dipacce Bono
public void makeSet(int node) {
    parent.put(node, node);
    rank.put(node, 0);
}

5 usages  Dipacce Bono
public int find(int node) {
    if (parent.get(node) != node) {
        parent.put(node, find(parent.get(node)));
    }
    return parent.get(node);
}

1 usage  Dipacce Bono
public void union(int node1, int node2) {
    int root1 = find(node1);
    int root2 = find(node2);

    if (root1 != root2) {
        if (rank.get(root1) > rank.get(root2)) {
            parent.put(root2, root1);
        } else if (rank.get(root1) < rank.get(root2)) {
            parent.put(root1, root2);
        } else {
            parent.put(root2, root1);
            rank.put(root1, rank.get(root1) + 1);
        }
    }
}
```

Esta clase permite determinar si dos nodos están en el mismo conjunto y unir conjuntos disjuntos de manera eficiente, evitando la formación de ciclos en el grafo.

`makeSet(int node)`:

- Inicializa un nodo, asignándose a sí mismo como su propio "padre".
- Establece el rango del nodo en 0.

`find(int node)`:

- Encuentra el "representante" o "raíz" del conjunto al que pertenece un nodo.

`union(int node1, int node2)`:

- Une dos conjuntos disjuntos usando la **unión por rango**, para optimizar la estructura de árboles.
- Actualiza el "padre" del nodo con menor rango al de mayor rango.

Resultado

Main

```java
import Estructura.*;
import javax.swing.*;

import static Estructura.KruskalAnimation.mst;


Dipacce Bono
public class Main {
    Dipacce Bono
    public static void main(String[] args) {
        GrafoDinamico grafo = new GrafoDinamico();
        // Agregar nodos y aristas al grafo
        grafo.agregarArista( origen: 1,  destino: 2,  peso: 3);
        grafo.agregarArista( origen: 1,  destino: 3,  peso: 2);
        grafo.agregarArista( origen: 1,  destino: 4,  peso: 3);
        grafo.agregarArista( origen: 2,  destino: 3,  peso: 1);
        grafo.agregarArista( origen: 2,  destino: 5,  peso: 5);
        grafo.agregarArista( origen: 3,  destino: 4,  peso: 2);
        grafo.agregarArista( origen: 3,  destino: 5,  peso: 7);
        grafo.agregarArista( origen: 3,  destino: 6,  peso: 8);
        grafo.agregarArista( origen: 4,  destino: 6,  peso: 6);
        grafo.agregarArista( origen: 5,  destino: 6,  peso: 4);

        JFrame frame = new JFrame( title: "Animación de Kruskal");
        KruskalAnimation animacion = new KruskalAnimation(grafo);
        frame.add(animacion);
        frame.pack();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);

        System.out.println("Arbol de Expansión Mínima (MST):");
        for (Arista arista : mst) {
            System.out.println("Nodo " + arista.getOrigen() + " - Nodo " + arista.getDestino() + " (Peso: " + arista.getPeso() + ")");
        }
    }
}
```

Consola

```
Arbol de Expansión Mínima (MST):
Nodo 3 - Nodo 2 (Peso: 1)
Nodo 1 - Nodo 3 (Peso: 2)
Nodo 4 - Nodo 3 (Peso: 2)
Nodo 6 - Nodo 5 (Peso: 4)
Nodo 2 - Nodo 5 (Peso: 5)
```

Demostración Gráfica