BFS

public static ResultadoBFS ejecutar(Grafo grafo, Localidad origen, Graph grafoVisual) {	
Map<Localidad, Integer> distancias = new HashMap<>();	1
Map<Localidad, Localidad> predecesores = new HashMap<>();	1
Queue<Localidad> cola = new LinkedList<>();	1
Set<Localidad> visitados = new HashSet<>();	1
distancias.put(origen, 0);	1
cola.add(origen);	1
visitados.add(origen);	1
System.out.println("Inicio en: " + origen.getNombre());	1
VisualizadorUtils.pintarNodo(grafoVisual, origen.getNombre(), "visitado");	
while (!cola.isEmpty()) {	V
Localidad actual = cola.poll();	V
System.out.println("Procesando: " + actual.getNombre());	V
for (Localidad vecino : grafo.obtenerVecinos(actual)) {	E
if (!visitados.contains(vecino)) {	E
distancias.put(vecino, distancias.get(actual) + 1);	E
predecesores.put(vecino, actual);	E
cola.add(vecino);	E
visitados.add(vecino);	E
System.out.println("Descubierto: " + vecino.getNombre() + " desde " + actual.getNombre());	1
VisualizadorUtils.pintarNodo(grafoVisual, vecino.getNombre(), "visitado");	
VisualizadorUtils.pintarArista(grafoVisual, actual.getNombre(), vecino.getNombre(), "seleccionada");	

try {
Thread.sleep(500);
} catch (InterruptedException e) {
Thread.currentThread().interrupt();
}
}
}
}	

return new ResultadoBFS(distancias, predecesores);	1
}	

T(n) = 8 + 3V + 6E
O(n) = O(V + E)

DFS

public static ResultadoDFS ejecutar(Grafo grafo, Graph visual) {	
visitados = new HashSet<>();	1
predecesores = new LinkedHashMap<>();	1
grafoVisual = visual;	1
// Visita cada componente del grafo	
for (Localidad u : grafo.getLocalidades()) {	V
if (!visitados.contains(u)) {	V
dfsVisitar(grafo, u);	V + E
}	

}
 	
return new ResultadoDFS(predecesores);
}
	
	1

private static void dfsVisitar(Grafo grafo, Localidad u) {
visitados.add(u);
System.out.println("Procesando: " + u.getNombre());
VisualizadorUtils.pintarNodo(grafoVisual, u.getNombre(), "visitado");
 	
for (Localidad v : grafo.obtenerVecinos(u)) {
if (!visitados.contains(v)) {
predecesores.put(v, u);
VisualizadorUtils.pintarArista(grafoVisual, u.getNombre(), v.getNombre(), "seleccionada");
 	
try {
Thread.sleep(500); // animación visual
} catch (InterruptedException e) {
Thread.currentThread().interrupt();
}
 	
dfsVisitar(grafo, v);
}
}
}
	
	1
	1
	
	E
	E
	E
	
	V + E

DFS
T(n) = 4 + 3V + E
O(n) = O(V + E)

dfsVisitar:
T(n) = 2 + V + 4E
O(n) = O(V + E)

Kruskal

public static ResultadoMST ejecutar(Grafo grafo, Graph grafoVisual) {	
List<Carretera> mst = new ArrayList<>();	1
List<Carretera> aristas = new ArrayList<>(grafo.getCarreteras());	E
aristas.sort(Comparator.comparingDouble(Carretera::getPeso));	E*logE
Map<Localidad, Localidad> padre = new HashMap<>();	1
for (Localidad loc : grafo.getLocalidades()) {	V
padre.put(loc, loc);	V
}	
for (Carretera c : aristas) {	E
Localidad a = c.getOrigen();	E
Localidad b = c.getDestino();	E
Localidad raizA = encontrar(padre, a);	E
Localidad raizB = encontrar(padre, b);	E
System.out.printf("Evaluando arista: %s - %s (%.2f km)%n", a.getNombre(), b.getNombre(), c.getPeso());	E
if (!raizA.equals(raizB)) {	V-1
padre.put(raizA, raizB);	V-1
mst.add(c);	V-1
System.out.printf("Arista agregada al MST: %s - %s%n", a.getNombre(), b.getNombre());	V-1
VisualizadorUtils.pintarNodo(grafoVisual, a.getNombre(), "visitado");	

VisualizadorUtils.pintarNodo(grafoVisual, b.getNombre(), "visitado");
VisualizadorUtils.pintarArista(grafoVisual, a.getNombre(), b.getNombre(), "seleccionada");
 	
try {
Thread.sleep(500);
} catch (InterruptedException e) {
Thread.currentThread().interrupt();
}
 	
if (mst.size() == grafo.getLocalidades().size() - 1) {
break;
}
} else {
System.out.printf("Arista descartada: %s - %s (formaría ciclo)%n", a.getNombre(), b.getNombre());
}
}
return new ResultadoMST(mst);
}
	
	n
	
	1
	
	1
	
	E-(V-1)
	
	1

T(V,E) = ElogE + 6E + 6V - 1
O(ElogE)

Prim

public static ResultadoMST ejecutar(Grafo grafo, Graph grafoVisual, Localidad inicio) { Map<Localidad, Double> key = new HashMap<>(); Map<Localidad, Carretera> aristaMasBarata = new HashMap<>(); Set<Localidad> enMST = new HashSet<>(); PriorityQueue<LocalidadAux> cola = new PriorityQueue<>();	 1 1 1 1

for (Localidad loc : grafo.getLocalidades()) {
  key.put(loc, Double.POSITIVE_INFINITY);
}
	V
	V

key.put(inicio, 0.0); cola.add(new LocalidadAux(inicio, 0.0));	1 logV
List<Carretera> mst = new ArrayList<>();	1
while (!cola.isEmpty()) { Localidad actual = cola.poll().localidad;	V VlogV
if (enMST.contains(actual)) continue; enMST.add(actual);	V V

if (aristaMasBarata.containsKey(actual)) {
  Carretera carretera = aristaMasBarata.get(actual);
  mst.add(carretera);
	V
	V-1
	V-1

VisualizadorUtils.pintarNodo(grafoVisual, carretera.getOrigen().getNombre(), "visitado");
VisualizadorUtils.pintarNodo(grafoVisual, carretera.getDestino().getNombre(), "visitado");
VisualizadorUtils.pintarArista(grafoVisual,
  carretera.getOrigen().getNombre(),
  carretera.getDestino().getNombre(),
  "seleccionada");	

System.out.println("Agregando arista: " + carretera.getOrigen().getNombre()
  + " - " + carretera.getDestino().getNombre()
  + " (" + carretera.getPeso() + " km)");
	V-1

try {
  Thread.sleep(500);
} catch (InterruptedException e) {
	V-1

Thread.currentThread().interrupt(); } }	1

for (Carretera c : grafo.getCarreteras()) {
Localidad vecino = null;
 	
if (c.getOrigen().equals(actual)) {
vecino = c.getDestino();
} else if (c.getDestino().equals(actual)) {
vecino = c.getOrigen();
}
 	
if (vecino != null && !enMST.contains(vecino) && c.getPeso() < key.get(vecino)) {
key.put(vecino, c.getPeso());
aristaMasBarata.put(vecino, c);
cola.add(new LocalidadAux(vecino, c.getPeso()));
}
}
}
	V*E
	V*E
	
	V*E
	
	V*E
	
	3*V*E
	V*E
	V*E
	V*E*logV

return new ResultadoMST(mst); }	1

T(V,E) = 9VE+VElogV+VlogV+logV+9V+3
O(VElogV)

Borůvka

public static ResultadoMST ejecutar(Grafo grafo, Graph grafoVisual) {	
List<Carretera> mst = new ArrayList<>();	1
List<Carretera> aristas = new ArrayList<>(grafo.getCarreteras());	E
Map<Localidad, Localidad> padre = new HashMap<>();	1
for (Localidad loc : grafo.getLocalidades()) {	V
padre.put(loc, loc);	
}	
int componentes = grafo.getLocalidades().size();	1
while (componentes > 1) {	logV
Map<Localidad, Carretera> masBarata = new HashMap<>();	logV
for (Carretera c : aristas) {	logV
Localidad a = c.getOrigen();	ElogV
Localidad b = c.getDestino();	ElogV
Localidad raizA = encontrar(padre, a);	ElogV
Localidad raizB = encontrar(padre, b);	ElogV
if (!raizA.equals(raizB)) {	
masBarata.putIfAbsent(raizA, c);	ElogV
masBarata.putIfAbsent(raizB, c);	ElogV
if (c.getPeso() < masBarata.get(raizA).getPeso()) {	ElogV
masBarata.put(raizA, c);	ElogV
}	
if (c.getPeso() < masBarata.get(raizB).getPeso()) {	ElogV
masBarata.put(raizB, c);	ElogV
}	
}	
}	

```
        for (Carretera c : new HashSet<>(masBarata.values())) {                                          V-1
            Localidad a = c.getOrigen();                                                                 V-1
            Localidad b = c.getDestino();                                                                V-1

            Localidad raizA = encontrar(padre, a);                                                       V-1
            Localidad raizB = encontrar(padre, b);                                                       V-1

            if (!raizA.equals(raizB)) {                                                                  V-1
                padre.put(raizA, raizB);                                                                 V-1
                mst.add(c);                                                                              V-1
                componentes--;                                                                           V-1

                VisualizadorUtils.pintarNodo(grafoVisual, a.getNombre(), "visitado");
                VisualizadorUtils.pintarNodo(grafoVisual, b.getNombre(), "visitado");
                VisualizadorUtils.pintarArista(grafoVisual, a.getNombre(), b.getNombre(), "seleccionada");

                System.out.println("Agregando arista: " + a.getNombre()                                  V-1
                        + " - " + b.getNombre()
                        + " (" + c.getPeso() + " km)");

                try {
                    Thread.sleep(500);                                                                   V-1
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                }
            }
        }
    }

    return new ResultadoMST(mst);
}                                                                                                        1
```

T(V,E) = 8ElogV+E+13V+3logV-8
O(ElogV)

Bellman - Ford

public static ResultadoCamino ejecutar(Grafo grafo, Localidad origen, Localidad destino, Graph grafoVisual) throws InterruptedException {	
Map<Localidad, Double> distancias = new HashMap<>();	1
Map<Localidad, Localidad> anteriores = new HashMap<>();	1
for (Localidad l : grafo.getLocalidades()) {	V
distancias.put(l, Double.POSITIVE_INFINITY);	V
}	
distancias.put(origen, 0.0);	1
int V = grafo.getLocalidades().size();	1
for (int i = 0; i < V - 1; i++) {	
for (Carretera c : grafo.getCarreteras()) {	V
Localidad u = c.getOrigen();	(E
Localidad v = c.getDestino();	E
double peso = c.getPeso();	E
	E
if (distancias.get(u) + peso < distancias.get(v)) {	
distancias.put(v, distancias.get(u) + peso);	E
anteriores.put(v, u);	E
}	E
if (distancias.get(v) + peso < distancias.get(u)) {	
distancias.put(u, distancias.get(v) + peso);	E
anteriores.put(u, v);	E

```
            }
        }
    }

    for (Carretera c : grafo.getCarreteras()) {
        Localidad u = c.getOrigen();
        Localidad v = c.getDestino();
        double peso = c.getPeso();

        if (distancias.get(u) + peso < distancias.get(v) || distancias.get(v) + peso < distancias.get(u)) {
            throw new RuntimeException("El grafo contiene un ciclo de peso negativo");
        }
    }

    List<Localidad> camino = new ArrayList<>();
    Localidad actual = destino;

    while (actual != null) {
        camino.add(0, actual);
        actual = anteriores.get(actual);
    }

    for (int i = 0; i < camino.size() - 1; i++) {
        Localidad u = camino.get(i);
        Localidad v = camino.get(i + 1);
        VisualizadorUtils.pintarArista(grafoVisual, u.getNombre(), v.getNombre(), "seleccionada");
        VisualizadorUtils.pintarNodo(grafoVisual, v.getNombre(), "destino");
        Thread.sleep(400);
    }
    VisualizadorUtils.pintarNodo(grafoVisual, origen.getNombre(), "origen");

    return new ResultadoCamino(camino, distancias.get(destino));
}
```

```
E)

E
E
E
E

E

1
1

V
V
V

1
```

T(n) = 10VE + 8V + 6E + 6
O(V * E + V)

Dijkstra

public static ResultadoCamino ejecutar(Grafo grafo, Localidad origen, Localidad destino, Graph grafoVisual) throws InterruptedException {	
Map<Localidad, Double> distancias = new HashMap<>();	1
Map<Localidad, Localidad> anteriores = new HashMap<>();	1
Set<Localidad> visitados = new HashSet<>();	1
PriorityQueue<Localidad> cola = new PriorityQueue<>(Comparator.comparingDouble(distancias::get));	1
for (Localidad l : grafo.getLocalidades()) {	V
distancias.put(l, Double.POSITIVE_INFINITY);	V
}	
distancias.put(origen, 0.0);	1
cola.add(origen);	1
while (!cola.isEmpty()) {	
Localidad actual = cola.poll();	V
	V
if (!visitados.add(actual)) {	
continue;	V
}	
VisualizadorUtils.pintarNodo(grafoVisual, actual.getNombre(), "visitado");	
Thread.sleep(300);	
if (actual.equals(destino)) {	V

```
            break;
        }

        for (Carretera carretera : grafo.getCarreteras()) {
            Localidad vecino = null;

            if (carretera.getOrigen().equals(actual)) {
                vecino = carretera.getDestino();
            } else if (carretera.getDestino().equals(actual)) {
                vecino = carretera.getOrigen();
            }

            if (vecino != null && !visitados.contains(vecino)) {
                double nuevaDistancia = distancias.get(actual) + carretera.getPeso();

                if (nuevaDistancia < distancias.get(vecino)) {
                    distancias.put(vecino, nuevaDistancia);
                    anteriores.put(vecino, actual);
                    cola.add(vecino);
                }
            }
        }
    }

    Localidad actual = destino;
    while (anteriores.containsKey(actual)) {
        Localidad anterior = anteriores.get(actual);

        VisualizadorUtils.pintarArista(grafoVisual, anterior.getNombre(), actual.getNombre(), "seleccionada");
        VisualizadorUtils.pintarNodo(grafoVisual, actual.getNombre(), "destino");

        actual = anterior;
        Thread.sleep(400);
    }
```

```
V*E
E


E
E
E
E


E
E


E
E
E
E


1
```

VisualizadorUtils.pintarNodo(grafoVisual, origen.getNombre(), "origen");	
List<Localidad> camino = new ArrayList<>();	1
actual = destino;	1
while (actual != null) {	V
camino.add(0, actual); // Insertar al inicio para mantener el or den correcto	V
actual = anteriores.get(actual);	V
}	
System.out.println("Distancia total del camino más corto: " + distancias.get(destino) + " km");	
return new ResultadoCamino(camino, distancias.get(destino));	
}	1

T(n) = 9VE + 8V + 11
O(VE)