**Complejidad computacional:**

El algoritmo utilizado para resolver el problema fue Dijkstra, el cual implementamos con cola de prioridad, motivo por el cual su complejidad computacional es de O(Alog(N)) siendo A la cantidad de aristas, y N la cantidad de nodos.

**public** **void** **dijkstra**() **throws** **IOException** {

**int** **nodoInicial** = 0;

// cola de prioridad para almacenar los nodos adyacentes que se van

// visitando

Queue<CostoAlNodo> **cola** = **new** PriorityQueue<CostoAlNodo>();

**CostoAlNodo** **nodoActual** = **new** CostoAlNodo(nodoInicial, 0, **this**.arboles.get(0));

// acolo el nodo inicial

cola.add(nodoActual);

**double** **peso** = 0;

**int** **nodo** = nodoActual.getNodo();

**CostoAlNodo** **actualizado**;

// inicializo la lista de costos

**for** (**int** **i** = 0; i < **this**.cantNodos; i++) {

**if** (i != nodo) {

costos.add(**new** CostoAlNodo(i, ***INFINITO***, **this**.arboles.get(i)));

} **else** {

costos.add(**new** CostoAlNodo(i, 0, **this**.arboles.get(i)));

costos.get(i).agregarNodoAlCamino(nodo);

}

}

// mientras la cola no está vacía

**while** (!cola.isEmpty()) {

nodoActual = cola.poll(); // desacolo al primero

nodo = nodoActual.getNodo();

// genero todos los números de nodo

**for** (**int** **i** = 0; i < **this**.cantNodos; i++) {

// si el nodo generado es distinto al nodo actual y no está

// terminado

**if** (nodo != i && !**this**.nodoTerminado[i]) {

// si hay arista que una los nodos

**if** ((peso = **this**.matAdy[nodo][i]) != ***INFINITO***) {

// si el nodo que estoy visitando no fue visitado o su

// costo mínimo es mayor que la distancia recorrida para

// visitarlo

**if** (**this**.costos.get(i).getCostoMinimo() == ***INFINITO*** || **this**.costos.get(nodo).getCostoMinimo()

+ peso < **this**.costos.get(i).getCostoMinimo()) {

// referencio al nodo

actualizado = **this**.costos.get(i);

// actualizo su costo al dado hasta el nodo actual

// más el peso de la arista entre ambos

actualizado.setCostoMinimo(**this**.costos.get(nodo).getCostoMinimo() + peso);

// le cambio el camino más corto por el camino más

// corto hasta el nodo actual

actualizado.setCaminoMasCorto(

(**ArrayList**<Integer>) **this**.costos.get(nodo).getCaminoMasCorto().clone());

// agrego el nodo visitado

actualizado.agregarNodoAlCamino(i);

// acolo el nodo visitado para luego buscar caminos

// desde él

**if** (!cola.contains(actualizado)) {

cola.add(actualizado);

}

}

}

}

}

// marco el nodo como terminado ya que su costo mínimo y camino más

// corto fueron hallados

**this**.nodoTerminado[nodo] = **true**;

}

}