ALGORITMOS VORACES

ETSII - URJC

Sebastián Ramiro Entierros García,
Adrián Martín Martín y
Álvaro Martínez Quiroga

Fundamentos del diseño del software

Índice

Contenido

Introducción y ficheros

En esta memoria se va a desarrollar la tercera práctica de la asignatura de *Fundamentos del Diseño del Software* del grado de *Ingeniería de Computadores* de la Universidad Rey Juan Carlos. La práctica consiste en el estudio de los algoritmos voraces a través del juego de mesa conocido como "Aventureros al tren - Europa". El objetivo de esta práctica es desarrollar dos algoritmos, estos son, uno que sea capaz de obtener el árbol de recubrimiento mínimo y otro que sea capaz de obtener el árbol de recubrimiento máximo. También responderemos a unas cuestiones comunes para este tipo de algoritmos y finalizaremos esta memoria con las conclusiones personales sobre el proyecto.

El paquete entregado, con nombre "Practica3_FDS", incluye los siguientes ficheros y archivos:

- Ficheros de código Java del programa.
- Memoria PDF.

Descripción del algoritmo

Como hemos mencionado, el algoritmo trata de obtener el camino de recubrimiento mínimo y el máximo, esto es:

- Camino mínimo: que el algoritmo sea capaz de unir todos los nodos haciendo uso únicamente de las aristas de menor coste.
- Camino máximo: que el algoritmo sea capaz de unir todos los nodos haciendo uso únicamente de las aristas de mayor coste.

Para ambos algoritmos hemos decidido **implementar el algoritmo de Kruskal** para el problema, ya que asegura una solución óptima de este.

Preguntas cortas

A continuación, responderemos a las preguntas presentadas para los algoritmos.

Camino mínimo

- Grupo de candidatos: las aristas del grafo.
- Función de solución: si la lista de nodos incluidos en la solución contiene todos los nodos del grafo.
- Función de factibilidad: siendo X una arista cualquiera del grafo, X se podrá incorporar a la solución solo si ambos, origen de X y destino de X, no están incluidos en la lista de solución.
- Función objetivo: obtener un subconjunto de aristas que, formando un grafo, este incluya todos los vértices (nodos) y donde el valor de coste de las aristas es el mínimo.
- Función de selección: elegir en cada iteración al candidato de menor coste.

Camino máximo

- Grupo de candidatos: las aristas del grafo.
- Función de solución: si la lista de nodos incluidos en la solución contiene todos los nodos del grafo.
- Función de factibilidad: siendo X una arista cualquiera del grafo, X se podrá incorporar a la solución solo si ambos, origen de X y destino de X, no están incluidos en la lista de solución.
- Función objetivo: obtener un subconjunto de aristas que, formando un grafo, este incluya todos los vértices (nodos) y donde el valor de coste de las aristas es el máximo.
- Función de selección: elegir en cada iteración al candidato de mayor coste.

Árbol de recubrimiento mínimo – Península Ibérica

Como hemos explicado con anterioridad, uno de los objetivos es conseguir el árbol de recubrimiento mínimo del grafo. Por simplicidad, explicaremos los ejemplos con el mapa de la península Ibérica.

Código camino mínimo

```
	public void calcMinSpanningTree() {
		int nodesToFill = this.nodes.size();
		System.out.println("\n--> INITIAL MIN TO MAX NODES KRUSKAL: " +
nodesToFill);

		// Pre-inicio: Creamos el subset
		// En este momento cada set del subset sera de solo 1 elemento
		SubsetsManager sbm = new SubsetsManager(this.nodes);
		// System.out.println(sbm.toString()); // Para comprobar los subsets

		// Primero Ordenamos los candidatos del problema
		// de coste menor a coste mayor
		ArrayList<Edge> temp_edges = this.edges;
		Collections.sort(temp_edges, new EdgeComparator());

		// Para comprobar la ordenacion de nodos
		/*
		 * for (Edge e: nodeList) { System.out.println(e.getSource() + "-" +
		 * e.getDestination() + ": " + e.getLength()); } System.out.println();
		 */

		int actualEdge = 0;
		int totalCost = 0;
		int totalEdges = 0;

		// Repetimos la iteracion si el numero de aristas
		// incluidos en la solucion es menor al numero de nodos - 1
		while (totalEdges < nodesToFill - 1 && // mientras no haya incluido ya
todos los nodos
				actualEdge < temp_edges.size() // mientras me queden nodos
		) {
			// Seleccionamos el nodo actual sobre el que operar
			Edge e = temp_edges.get(actualEdge);

			// Comprobamos si incluir ese nodo en la solucion
			// genera o no un bucle en el arbol solucion
			if (!sbm.findSourceAndDestinationInSubsets(e)) {
				// Si no hay ningun subset que contenga
				// el destino y el origen, el nodo es valido
				System.out.println("Inlcuyendo nodo: " + e.getSource() + "-"
+ e.getDestination() + " con coste: "
						+ e.getLength() + " en la solucion");
				sbm.updateSubsetsWithEdge(e);
				totalEdges++;
				totalCost += e.getLength();
			} else {
				System.out.println("No inlcuyendo nodo: " + e.getSource() +
"-" + e.getDestination() + " con coste: "
						+ e.getLength() + " no es factible para la
solucion, genera un bucle en el grafo");
			}
			actualEdge++;
```

```
            }

            System.out.println("\n" + sbm.toString() + "Coste: " + totalCost); //
Imprimimos los subsets
        }
```

Descripción

El algoritmo es sencillo y sigue una serie de pasos para conseguir su objetivo, estos son:

1. Hacemos uso de la clase “SubSetManager” para crear tantas entradas como nodos tenga el problema. Este sub-set irá disminuyendo a medida que el algoritmo se ejecuta.
2. Ordenamos los candidatos del problema, estos son, los nodos y sus aristas. Se ordena de **menor a mayor** apoyándonos en los pesos de las aristas.
3. Comenzamos la ejecución del bucle del algoritmo, este terminará si se dan dos condiciones, que son:
    a. Mientras que no se hayan incluido ya todos los nodos del problema en la solución.
    b. Mientras el algoritmo tenga nodos disponibles.
4. El nodo X a tratar en una iteración Y del bucle será factible para añadir a la solución solo si el origen y destino de este nodo no se han incluido ya en la solución de sub-sets. No comprobamos el peso de las aristas en este momento ya que han de estar ordenadas antes de empezar el bucle.
5. Finalmente imprimimos el resultado.

Ejemplo por consola

```
--> INITIAL MIN TO MAX NODES KRUSKAL: 5
Inlcuyendo nodo: Madrid-Barcelona con coste: 2 en la solucion
Inlcuyendo nodo: Lisboa-Cadiz con coste: 2 en la solucion
Inlcuyendo nodo: Madrid-Lisboa con coste: 3 en la solucion
No inlcuyendo nodo: Madrid-Cadiz con coste: 3 no es factible para la solucion,
genera un bucle en el grafo
Inlcuyendo nodo: Barcelona-Pamplona con coste: 4 en la solucion

Subset: Madrid-Barcelona-Lisboa-Cadiz-Pamplona
Coste: 11
```

Como podemos observar por la salida de la ejecución del problema, comienza presentando el número de nodos a tratar. También, excluye las aristas que generan bules en el grafo solución. Finalmente, cuando ha recorrido todos los nodos, termina el bucle e imprime el sub-set del orden de inclusión de nodos junto con el coste de todas estas inclusiones en la lista solución. Finalmente **genera un coste del valor del peso de las aristas igual a 11**.

Árbol de recubrimiento máximo – Península Ibérica

El segundo objetivo es conseguir el árbol de recubrimiento máximo, como en el anterior árbol utilizaremos el ejemplo de la península ibérica para simplificar. El código respecto a este método es muy similar al anterior, cambiando únicamente el titulo y la manera de ordenar el vector.

Código camino máximo

```
	public void calcMinSpanningTree() {
		int nodesToFill = this.nodes.size();
		System.out.println("\n--> INITIAL MAX TO MIN NODES KRUSKAL: " +
nodesToFill);

		// Pre-inicio: Creamos el subset
		// En este momento cada set del subset sera de solo 1 elemento
		SubsetsManager sbm = new SubsetsManager(this.nodes);
		// System.out.println(sbm.toString()); // Para comprobar los subsets

		// Primero Ordenamos los candidatos del problema
		// de coste mayor a menor
		ArrayList<Edge> temp_edges = this.edges;
		Collections.sort(temp_edges, new EdgeComparatorInverse());

		// Para comprobar la ordenacion de nodos
		/*
		 * for (Edge e: nodeList) { System.out.println(e.getSource() + "-" +
		 * e.getDestination() + ": " + e.getLength()); } System.out.println();
		 */

		int actualEdge = 0;
		int totalCost = 0;
		int totalEdges = 0;

		// Repetimos la iteracion si el numero de aristas
		// incluidos en la solucion es menor al numero de nodos - 1
		while (totalEdges < nodesToFill - 1 && // mientras no haya incluido ya
todos los nodos
				actualEdge < temp_edges.size() // mientras me queden nodos
		) {
			// Seleccionamos el nodo actual sobre el que operar
			Edge e = temp_edges.get(actualEdge);

			// Comprobamos si incluir ese nodo en la solucion
			// genera o no un bucle en el arbol solucion
			if (!sbm.findSourceAndDestinationInSubsets(e)) {
				// Si no hay ningun subset que contenga
				// el destino y el origen, el nodo es valido
				System.out.println("Inlcuyendo nodo: " + e.getSource() + "-"
+ e.getDestination() + " con coste: "
						+ e.getLength() + " en la solucion");
				sbm.updateSubsetsWithEdge(e);
				totalEdges++;
				totalCost += e.getLength();
			} else {
				System.out.println("No inlcuyendo nodo: " + e.getSource() +
"-" + e.getDestination() + " con coste: "
						+ e.getLength() + " no es factible para la
solucion, genera un bucle en el grafo");
			}
```

```
                    actualEdge++;
                }

                System.out.println("\n" + sbm.toString() + "Coste: " + totalCost); //
Imprimimos los subsets
        }
```

Descripción

El algoritmo es sencillo y sigue una serie de pasos para conseguir su objetivo, estos son:

1. Hacemos uso de la clase “SubSetManager” para crear tantas entradas como nodos tenga el problema. Este sub-set irá disminuyendo a medida que el algoritmo se ejecuta.
2. Ordenamos los candidatos del problema, estos son, los nodos y sus aristas. Se ordena de **mayor a menor** apoyándonos en los pesos de las aristas.
3. Comenzamos la ejecución del bucle del algoritmo, este terminará si se dan dos condiciones, que son:
    a. Mientras que no se hayan incluido ya todos los nodos del problema en la solución.
    b. Mientras el algoritmo tenga nodos disponibles.
4. El nodo X a tratar en una iteración Y del bucle será factible para añadir a la solución solo si el origen y destino de este nodo no se han incluido ya en la solución de sub-sets. No comprobamos el peso de las aristas en este momento ya que han de estar ordenadas antes de empezar el bucle.
5. Finalmente imprimimos el resultado.

Ejemplo por consola

```
--> INITIAL MAX TO MIN NODES KRUSKAL: 5
Incluyendo nodo: Madrid-Pamplona con coste: 6 en la solución
Incluyendo nodo: Barcelona-Pamplona con coste: 4 en la solución
Incluyendo nodo: Madrid-Lisboa con coste: 3 en la solución
Incluyendo nodo: Madrid-Cadiz con coste: 3 en la solución

Subset: Barcelona-Madrid-Pamplona-Lisboa-Cadiz-
Coste: 16
```

Como podemos observar por la salida de la ejecución del problema, comienza presentando el número de nodos a tratar. También, excluye las aristas que generan bules en el grafo solución (en este ejemplo no ha hecho falta). Finalmente, cuando ha recorrido todos los nodos, termina el bucle e imprime el sub-set del orden de inclusión de nodos junto con el coste de todas estas inclusiones en la lista solución. Finalmente **genera un coste del valor del peso de las aristas igual a 16**.

Caso completo – Europa

En este punto mostraremos todas las aristas y el resultado de los algoritmos para el caso completo (mapa de todo Europa) del problema.

Tabla de aristas recorrido mínimo

Origen	Destino	Coste
Paris	Dieppe	1
Budapest	Viena	1
Amsterdam	Bruselas	1
Madrid	Barcelona	2
Lisboa	Cadiz	2
Paris	Bruselas	2
Roma	Venecia	2
Roma	Brindisi	2
Venecia	Zagreb	2
Zagreb	Budapest	2
Atenas	Esmirna	2
Sebastopol	Sochi	2
Sochi	Rostov	2
Rostov	Jarkov	2
Kiev	Vilna	2
Smolensk	Moscu	2
Varsovia	Danzig	2
Munich	Frankfurt	2
Berlin	Essen	2
Frankfurt	Essen	2
Frankfurt	Amsterdam	2
Amsterdam	Londres	2
Dieppe	Brest	2
Madrid	Lisboa	3
Zagreb	Sarajevo	3
Brindisi	Palermo	3
Atenas	Sofia	3
Sofia	Estambul	3
Bucarest	Estambul	3
Ankara	Erzurum	3
Kiev	Smolensk	3
Vilna	Varsovia	3
Riga	Danzig	3
Estocolmo	Copenhague	3
Viena	Munich	3
Essen	Copenhague	3
Barcelona	Pamplona	4

Barcelona	Marsella	4
Pamplona	Paris	4
Zurich	Marsella	4
Brindisi	Atenas	4
Bucarest	Sebastopol	4
Bucarest	Kiev	4
Sebastopol	Erzurum	4
Moscu	San Petersburgo	4
Londres	Edimburgo	4

Ejemplo por consola recorrido mínimo:

```
--> INITIAL MIN TO MAX NODES KRUSKAL: 47

Incluyendo nodo: Paris-Dieppe con coste: 1 en la solución
Incluyendo nodo: Budapest-Viena con coste: 1 en la solución
Incluyendo nodo: Amsterdan-Bruselas con coste: 1 en la solución
Incluyendo nodo: Madrid-Barcelona con coste: 2 en la solución
Incluyendo nodo: Lisboa-Cadiz con coste: 2 en la solución
Incluyendo nodo: Paris-Bruselas con coste: 2 en la solución
Incluyendo nodo: Roma-Venecia con coste: 2 en la solución
Incluyendo nodo: Roma-Brindisi con coste: 2 en la solución
Incluyendo nodo: Venecia-Zagreb con coste: 2 en la solución
Incluyendo nodo: Zagreb-Budapest con coste: 2 en la solución
No incluyendo nodo: Zagreb-Viena con coste: 2 no es factible para la solución,
genera un bucle en el grafo
Incluyendo nodo: Atenas-Esmirna con coste: 2 en la solución
Incluyendo nodo: Sebastopol-Sochi con coste: 2 en la solución
Incluyendo nodo: Sochi-Rostov con coste: 2 en la solución
Incluyendo nodo: Rostov-Jarkov con coste: 2 en la solución
Incluyendo nodo: Kiev-Vilna con coste: 2 en la solución
Incluyendo nodo: Smolensk-Moscu con coste: 2 en la solución
Incluyendo nodo: Varsovia-Danzig con coste: 2 en la solución
Incluyendo nodo: Munich-Frankfurt con coste: 2 en la solución
Incluyendo nodo: Berlin-Essen con coste: 2 en la solución
Incluyendo nodo: Frankfurt-Essen con coste: 2 en la solución
Incluyendo nodo: Frankfurt-Amsterdan con coste: 2 en la solución
No incluyendo nodo: Frankfurt-Bruselas con coste: 2 no es factible para la
solución, genera un bucle en el grafo
Incluyendo nodo: Amsterdan-Londres con coste: 2 en la solución
No incluyendo nodo: Bruselas-Dieppe con coste: 2 no es factible para la
solución, genera un bucle en el grafo
No incluyendo nodo: Londres-Dieppe con coste: 2 no es factible para la solución,
genera un bucle en el grafo
Incluyendo nodo: Dieppe-Brest con coste: 2 en la solución
Incluyendo nodo: Madrid-Lisboa con coste: 3 en la solución
No incluyendo nodo: Madrid-Cadiz con coste: 3 no es factible para la solución,
genera un bucle en el grafo
No incluyendo nodo: Paris-Frankfurt con coste: 3 no es factible para la
solución, genera un bucle en el grafo
No incluyendo nodo: Paris-Brest con coste: 3 no es factible para la solución,
genera un bucle en el grafo
Incluyendo nodo: Zagreb-Sarajevo con coste: 3 en la solución
```

```
Incluyendo nodo: Brindisi-Palermo con coste: 3 en la solución
No incluyendo nodo: Sarajevo-Budapest con coste: 3 no es factible para la
solución, genera un bucle en el grafo
Incluyendo nodo: Atenas-Sofia con coste: 3 en la solución
Incluyendo nodo: Sofia-Estambul con coste: 3 en la solución
Incluyendo nodo: Bucarest-Estambul con coste: 3 en la solución
Incluyendo nodo: Ankara-Erzurum con coste: 3 en la solución
Incluyendo nodo: Kiev-Smolensk con coste: 3 en la solución
No incluyendo nodo: Smolensk-Vilna con coste: 3 no es factible para la solución,
genera un bucle en el grafo
Incluyendo nodo: Vilna-Varsovia con coste: 3 en la solución
Incluyendo nodo: Riga-Danzig con coste: 3 en la solución
Incluyendo nodo: Estocolmo-Copenhague con coste: 3 en la solución
Incluyendo nodo: Viena-Munich con coste: 3 en la solución
No incluyendo nodo: Viena-Berlin con coste: 3 no es factible para la solución,
genera un bucle en el grafo
No incluyendo nodo: Berlin-Frankfurt con coste: 3 no es factible para la
solución, genera un bucle en el grafo
Incluyendo nodo: Essen-Copenhague con coste: 3 en la solución
No incluyendo nodo: Essen-Amsterdan con coste: 3 no es factible para la
solución, genera un bucle en el grafo
Incluyendo nodo: Barcelona-Pamplona con coste: 4 en la solución
Incluyendo nodo: Barcelona-Marsella con coste: 4 en la solución
Incluyendo nodo: Pamplona-Paris con coste: 4 en la solución
No incluyendo nodo: Pamplona-Brest con coste: 4 no es factible para la solución,
genera un bucle en el grafo
Incluyendo nodo: Zurich-Marsella con coste: 4 en la solución
No incluyendo nodo: Zurich-Venecia con coste: 4 no es factible para la solución,
genera un bucle en el grafo
No incluyendo nodo: Zurich-Munich con coste: 4 no es factible para la solución,
genera un bucle en el grafo
No incluyendo nodo: Roma-Palermo con coste: 4 no es factible para la solución,
genera un bucle en el grafo
No incluyendo nodo: Venecia-Munich con coste: 4 no es factible para la solución,
genera un bucle en el grafo
Incluyendo nodo: Brindisi-Atenas con coste: 4 en la solución
No incluyendo nodo: Sarajevo-Atenas con coste: 4 no es factible para la
solución, genera un bucle en el grafo
No incluyendo nodo: Sarajevo-Sofia con coste: 4 no es factible para la solución,
genera un bucle en el grafo
No incluyendo nodo: Sofia-Bucarest con coste: 4 no es factible para la solución,
genera un bucle en el grafo
Incluyendo nodo: Bucarest-Sebastopol con coste: 4 en la solución
Incluyendo nodo: Bucarest-Kiev con coste: 4 en la solución
No incluyendo nodo: Sebastopol-Estambul con coste: 4 no es factible para la
solución, genera un bucle en el grafo
Incluyendo nodo: Sebastopol-Erzurum con coste: 4 en la solución
No incluyendo nodo: Sebastopol-Rostov con coste: 4 no es factible para la
solución, genera un bucle en el grafo
No incluyendo nodo: Estambul-Ankara con coste: 4 no es factible para la
solución, genera un bucle en el grafo
No incluyendo nodo: Estambul-Esmirna con coste: 4 no es factible para la
solución, genera un bucle en el grafo
No incluyendo nodo: Jarkov-Kiev con coste: 4 no es factible para la solución,
genera un bucle en el grafo
No incluyendo nodo: Kiev-Varsovia con coste: 4 no es factible para la solución,
genera un bucle en el grafo
```

```
Incluyendo nodo: Moscu-San Petersburgo con coste: 4 en la solución
No incluyendo nodo: Vilna-San Petersburgo con coste: 4 no es factible para la
solución, genera un bucle en el grafo
No incluyendo nodo: Vilna-Riga con coste: 4 no es factible para la solución,
genera un bucle en el grafo
No incluyendo nodo: Varsovia-Viena con coste: 4 no es factible para la solución,
genera un bucle en el grafo
No incluyendo nodo: Varsovia-Berlin con coste: 4 no es factible para la
solución, genera un bucle en el grafo
No incluyendo nodo: San Petersburgo-Riga con coste: 4 no es factible para la
solución, genera un bucle en el grafo
No incluyendo nodo: Danzig-Berlin con coste: 4 no es factible para la solución,
genera un bucle en el grafo
Incluyendo nodo: Londres-Edimburgo con coste: 4 en la solución

Subset: Zurich-Madrid-Barcelona-Lisboa-Cadiz-Pamplona-Marsella-Roma-Venecia-
Brindisi-Zagreb-Budapest-Viena-Sarajevo-Palermo-Munich-Frankfurt-Berlin-Essen-
Paris-Dieppe-Amsterdan-Bruselas-Londres-Brest-Estocolmo-Copenhague-Bucarest-
Atenas-Esmirna-Sofia-Estambul-Sebastopol-Sochi-Rostov-Jarkov-Riga-Kiev-Vilna-
Smolensk-Moscu-Varsovia-Danzig-Ankara-Erzurum-San Petersburgo-Edimburgo
Coste: 122
```

Podemos observar aquellos nodos no incluidos durante las iteraciones por la generación de bucles en grafo. Finalmente **genera un coste del valor del peso de las aristas igual a 122**.

Tabla de aristas recorrido máximo

Origen	Destino	Coste
San Petersburgo	Estocolmo	16
Budapest	Kiev	12
Marsella	Roma	8
Budapest	Bucarest	8
Madrid	Pamplona	6
Paris	Zurich	6
Palermo	Esmirna	6
Ankara	Esmirna	6
Erzurum	Sochi	6
Barcelona	Pamplona	4
Barcelona	Marsella	4
Pamplona	Paris	4
Pamplona	Brest	4
Zurich	Venecia	4
Zurich	Munich	4
Roma	Palermo	4
Brindisi	Atenas	4
Sarajevo	Atenas	4
Sarajevo	Sofia	4
Sofia	Bucarest	4

Bucarest	Sebastopol	4
Sebastopol	Estambul	4
Sebastopol	Erzurum	4
Sebastopol	Rostov	4
Estambul	Ankara	4
Jarkov	Kiev	4
Kiev	Varsovia	4
Moscu	San Petersburgo	4
Vilna	San Petersburgo	4
Vilna	Riga	4
Varsovia	Viena	4
Varsovia	Berlin	4
Danzig	Berlin	4
Londres	Edimburgo	4
Madrid	Lisboa	3
Madrid	Cadiz	3
Paris	Frankfurt	3
Zagreb	Sarajevo	3
Kiev	Smolensk	3
Smolensk	Vilna	3
Estocolmo	Copenhague	3
Essen	Copenhague	3
Essen	Amsterdam	3
Paris	Bruselas	2
Amsterdam	Londres	2
Bruselas	Dieppe	2

Ejemplo por consola recorrido máximo:

```
--> INITIAL MAX TO MIN NODES KRUSKAL: 47

Incluyendo nodo: San Petersburgo-Estocolmo con coste: 16 en la solución
Incluyendo nodo: Budapest-Kiev con coste: 12 en la solución
Incluyendo nodo: Marsella-Roma con coste: 8 en la solución
Incluyendo nodo: Budapest-Bucarest con coste: 8 en la solución
Incluyendo nodo: Madrid-Pamplona con coste: 6 en la solución
Incluyendo nodo: París-Zurich con coste: 6 en la solución
Incluyendo nodo: Palermo-Esmirna con coste: 6 en la solución
Incluyendo nodo: Ankara-Esmirna con coste: 6 en la solución
Incluyendo nodo: Erzurum-Sochi con coste: 6 en la solución
Incluyendo nodo: Barcelona-Pamplona con coste: 4 en la solución
Incluyendo nodo: Barcelona-Marsella con coste: 4 en la solución
Incluyendo nodo: Pamplona-París con coste: 4 en la solución
Incluyendo nodo: Pamplona-Brest con coste: 4 en la solución
No incluyendo nodo: Zurich-Marsella con coste: 4 no es factible para la
solución, genera un bucle en el grafo
Incluyendo nodo: Zurich-Venecia con coste: 4 en la solución
Incluyendo nodo: Zurich-Munich con coste: 4 en la solución
Incluyendo nodo: Roma-Palermo con coste: 4 en la solución
```

```
No incluyendo nodo: Venecia-Munich con coste: 4 no es factible para la solución,
genera un bucle en el grafo
Incluyendo nodo: Brindisi-Atenas con coste: 4 en la solución
Incluyendo nodo: Sarajevo-Atenas con coste: 4 en la solución
Incluyendo nodo: Sarajevo-Sofia con coste: 4 en la solución
Incluyendo nodo: Sofia-Bucarest con coste: 4 en la solución
Incluyendo nodo: Bucarest-Sebastopol con coste: 4 en la solución
No incluyendo nodo: Bucarest-Kiev con coste: 4 no es factible para la solución,
genera un bucle en el grafo
Incluyendo nodo: Sebastopol-Estambul con coste: 4 en la solución
Incluyendo nodo: Sebastopol-Erzurum con coste: 4 en la solución
Incluyendo nodo: Sebastopol-Rostov con coste: 4 en la solución
Incluyendo nodo: Estambul-Ankara con coste: 4 en la solución
No incluyendo nodo: Estambul-Esmirna con coste: 4 no es factible para la
solución, genera un bucle en el grafo
Incluyendo nodo: Jarkov-Kiev con coste: 4 en la solución
Incluyendo nodo: Kiev-Varsovia con coste: 4 en la solución
Incluyendo nodo: Moscú-San Petersburgo con coste: 4 en la solución
Incluyendo nodo: Vilna-San Petersburgo con coste: 4 en la solución
Incluyendo nodo: Vilna-Riga con coste: 4 en la solución
Incluyendo nodo: Varsovia-Viena con coste: 4 en la solución
Incluyendo nodo: Varsovia-Berlín con coste: 4 en la solución
No incluyendo nodo: San Petersburgo-Riga con coste: 4 no es factible para la
solución, genera un bucle en el grafo
Incluyendo nodo: Danzig-Berlín con coste: 4 en la solución
Incluyendo nodo: Londres-Edimburgo con coste: 4 en la solución
Incluyendo nodo: Madrid-Lisboa con coste: 3 en la solución
Incluyendo nodo: Madrid-Cádiz con coste: 3 en la solución
Incluyendo nodo: París-Frankfurt con coste: 3 en la solución
No incluyendo nodo: París-Brest con coste: 3 no es factible para la solución,
genera un bucle en el grafo
Incluyendo nodo: Zagreb-Sarajevo con coste: 3 en la solución
No incluyendo nodo: Brindisi-Palermo con coste: 3 no es factible para la
solución, genera un bucle en el grafo
No incluyendo nodo: Sarajevo-Budapest con coste: 3 no es factible para la
solución, genera un bucle en el grafo
No incluyendo nodo: Atenas-Sofia con coste: 3 no es factible para la solución,
genera un bucle en el grafo
No incluyendo nodo: Sofia-Estambul con coste: 3 no es factible para la solución,
genera un bucle en el grafo
No incluyendo nodo: Bucarest-Estambul con coste: 3 no es factible para la
solución, genera un bucle en el grafo
No incluyendo nodo: Ankara-Erzurum con coste: 3 no es factible para la solución,
genera un bucle en el grafo
Incluyendo nodo: Kiev-Smolensk con coste: 3 en la solución
Incluyendo nodo: Smolensk-Vilna con coste: 3 en la solución
No incluyendo nodo: Vilna-Varsovia con coste: 3 no es factible para la solución,
genera un bucle en el grafo
No incluyendo nodo: Riga-Danzig con coste: 3 no es factible para la solución,
genera un bucle en el grafo
Incluyendo nodo: Estocolmo-Copenhague con coste: 3 en la solución
No incluyendo nodo: Viena-Munich con coste: 3 no es factible para la solución,
genera un bucle en el grafo
No incluyendo nodo: Viena-Berlín con coste: 3 no es factible para la solución,
genera un bucle en el grafo
No incluyendo nodo: Berlín-Frankfurt con coste: 3 no es factible para la
solución, genera un bucle en el grafo
```

```
Incluyendo nodo: Essen-Copenhague con coste: 3 en la solución
Incluyendo nodo: Essen-Amsterdan con coste: 3 en la solución
No incluyendo nodo: Madrid-Barcelona con coste: 2 no es factible para la
solución, genera un bucle en el grafo
No incluyendo nodo: Lisboa-Cádiz con coste: 2 no es factible para la solución,
genera un bucle en el grafo
Incluyendo nodo: París-Bruselas con coste: 2 en la solución
No incluyendo nodo: Roma-Venecia con coste: 2 no es factible para la solución,
genera un bucle en el grafo
No incluyendo nodo: Roma-Brindisi con coste: 2 no es factible para la solución,
genera un bucle en el grafo
No incluyendo nodo: Venecia-Zagreb con coste: 2 no es factible para la solución,
genera un bucle en el grafo
No incluyendo nodo: Zagreb-Budapest con coste: 2 no es factible para la
solución, genera un bucle en el grafo
No incluyendo nodo: Zagreb-Viena con coste: 2 no es factible para la solución,
genera un bucle en el grafo
No incluyendo nodo: Atenas-Esmirna con coste: 2 no es factible para la solución,
genera un bucle en el grafo
No incluyendo nodo: Sebastopol-Sochi con coste: 2 no es factible para la
solución, genera un bucle en el grafo
No incluyendo nodo: Sochi-Rostov con coste: 2 no es factible para la solución,
genera un bucle en el grafo
No incluyendo nodo: Rostov-Jarkov con coste: 2 no es factible para la solución,
genera un bucle en el grafo
No incluyendo nodo: Kiev-Vilna con coste: 2 no es factible para la solución,
genera un bucle en el grafo
No incluyendo nodo: Smolensk-Moscú con coste: 2 no es factible para la solución,
genera un bucle en el grafo
No incluyendo nodo: Varsovia-Danzig con coste: 2 no es factible para la
solución, genera un bucle en el grafo
No incluyendo nodo: Munich-Frankfurt con coste: 2 no es factible para la
solución, genera un bucle en el grafo
No incluyendo nodo: Berlín-Essen con coste: 2 no es factible para la solución,
genera un bucle en el grafo
No incluyendo nodo: Frankfurt-Essen con coste: 2 no es factible para la
solución, genera un bucle en el grafo
No incluyendo nodo: Frankfurt-Amsterdan con coste: 2 no es factible para la
solución, genera un bucle en el grafo
No incluyendo nodo: Frankfurt-Bruselas con coste: 2 no es factible para la
solución, genera un bucle en el grafo
Incluyendo nodo: Amsterdan-Londres con coste: 2 en la solución
Incluyendo nodo: Bruselas-Dieppe con coste: 2 en la solución

Subset: Essen-Zagreb-Danzig-Jarkov-Sarajevo-Brindisi-Atenas-Sofia-Budapest-Kiev-
Bucarest-Sebastopol-Estambul-Erzurum-Sochi-Rostov-Barcelona-Madrid-Pamplona-
Marsella-Roma-París-Zurich-Brest-Venecia-Munich-Ankara-Palermo-Esmirna-Varsovia-
Viena-Berlín-Lisboa-Cádiz-Frankfurt-Smolensk-Vilna-Moscú-San Petersburgo-
Estocolmo-Riga-Copenhague-Amsterdan-Bruselas-Londres-Edimburgo-Dieppe-
Coste: 207
```

Podemos observar aquellos nodos no incluidos durante las iteraciones por la generación de bucles en grafo. Finalmente **genera un coste del valor del peso de las aristas igual a 207**.

Conclusiones de la práctica.

Consideramos fundamental esta práctica para profundizar en el funcionamiento, desarrollo y uso de los algoritmos voraces. Al realizar ambos algoritmos hemos podido comprobar que la única diferencia entre ambos es la forma de ordenar el vector.

Al comienzo del desarrollo de la práctica tuvimos dificultades en entender el código proporcionado para esta, sin embargo, cuando ya nos adaptamos a este, nos dimos cuenta de lo esencial que era para el desarrollo de esta práctica. Cabe destacar que incluso hemos incluido los métodos “toString” en las clases “SubSetManager” y “SubSet”, para que así fuese más fácil la depuración del problema y así entender cómo funcionaban las clases de manera más clara.

Referencias

- Temario del aula virtual.
- Algoritmo de Kruskal.