

## MST

### Manual de Usuario

0. El siguiente manual explica cómo utilizar tres funciones que nos permiten encontrar el árbol de expansión mínimo de un grafo mediante los algoritmos de Prim, Kruskal usando búsqueda en profundidad y Kruskal usando la estructura de datos Union-Find. Estas funciones se realizaron a través de una clase en el lenguaje de programación Java. Para utilizar esta clase es necesario tener Java instalado en el equipo y colocar el archivo de texto con los datos de entrada dentro de la misma carpeta en donde se encuentre el archivo Java que acompaña a este manual. Esta API fue construida y probada utilizando Eclipse IDE.

Se mostrarán las funciones de las clases y algunos de sus ejemplos de uso.

1. **La clase MST**

Esta clase es la básica primaria que guardará todos las funciones de nuestra implementación.

Dentro de la función main que se utilice, será necesario declarar un objeto de la clase MST sobre el cual se realizarán las operaciones.

Ejemplo:

```
MST miMST=new MST();
```

Cada función lee el grafo del archivo que recibe como parámetro, obtiene su MST con el algoritmo dado y regresa su costo. Además imprime el tiempo de ejecución y las aristas del MST en el siguiente formato: (**nodoInicial, nodoFinal, costo**).

2. **Formato del archive de texto**

Para implementar las funciones que se mencionaran es necesario que el archivo de texto que se pasa como parámetro cumpla con las siguientes especificaciones que sirven para describir un grafo.

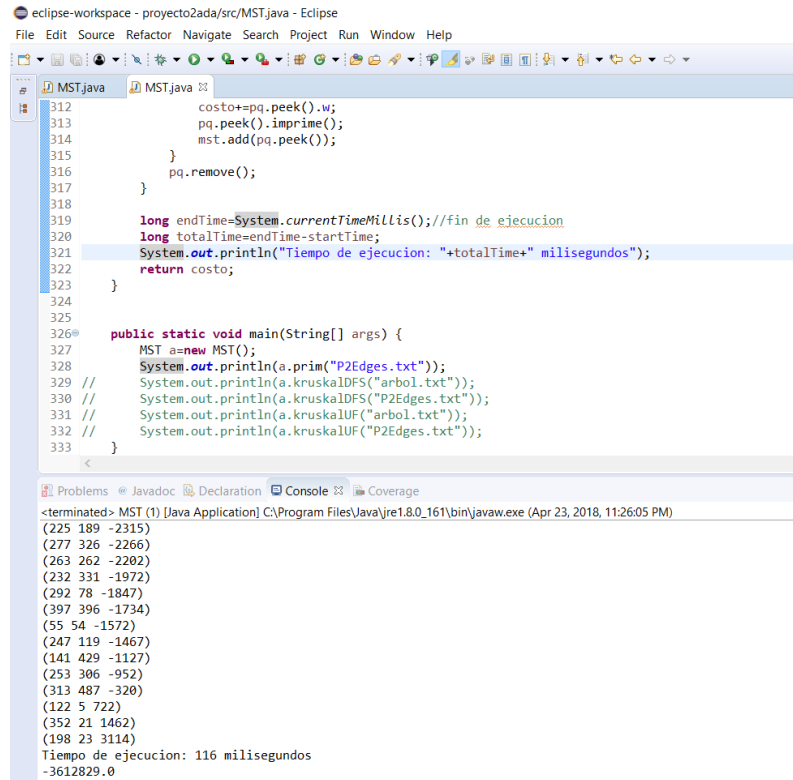
- i) La primera línea contiene dos enteros positivos **n** y **m**, el primero de ellos (**n**) corresponde a la cantidad de vértices en el grafo y el siguiente (**m**) corresponde a la cantidad de aristas.
- ii) Las siguientes **m** líneas consisten de tres enteros: **nodoInicial**, **nodoFinal** y **peso**, mismos que están separados por un espacio simple. Estos tres valores constituyen una arista del grafo. Es por ello que hay m líneas de este tipo.
- iii) Los valores para **nodoInicial** y **nodoFinal** deben ser enteros positivos en el conjunto [1, n] y cada número debe aparecer en algún momento ya sea en **nodoInicial** o en **nodoFinal**
- iv) El valor **peso** es un entero que puede ser negativo o cero

3. **Función float prim(String archivo)**

Recibe el archivo y por medio del algoritmo de Prim para encontrar el árbol de expansión mínima, regresa lo siguiente:

El valor del costo mínimo del árbol de expansión, mismo que se da como una variable float. Al mismo tiempo que da el valor del costo, esta función regresa las aristas una a una como se fueron insertando en el árbol de expansión mínimo.

Después, nuestra función muestra el tiempo que tardó en implementarse mediante una impresión en consola que indica los milisegundos que se tardó.



The screenshot shows the Eclipse IDE with a Java project named 'proyecto2ada'. The file 'MST.java' is open in the editor. The code implements a Minimum Spanning Tree (MST) algorithm. It includes a method to calculate the total time of execution and a main method that tests the algorithm on a graph with 20 vertices and 25 edges. The console output shows the execution time and the total cost of the MST.

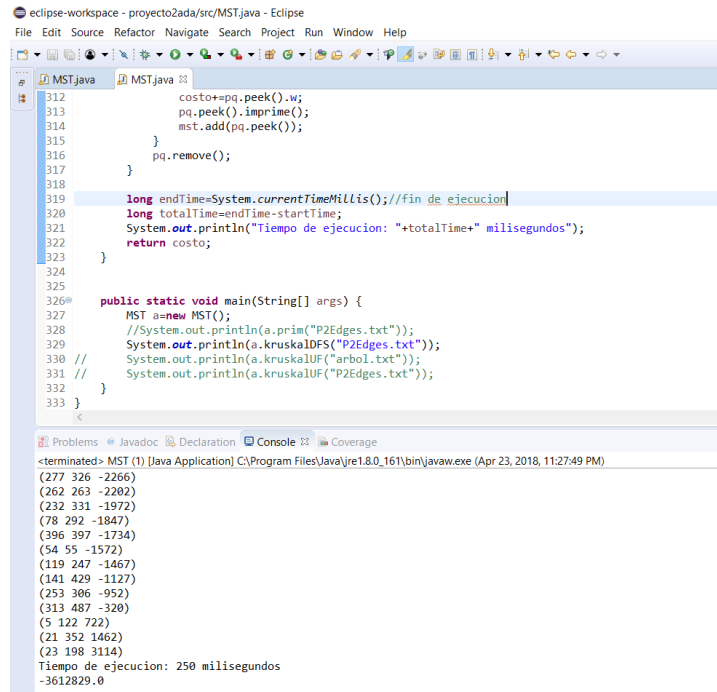
```
312         costo+=pq.peek().w;
313         pq.peek().imprime();
314         mst.add(pq.peek());
315     }
316     pq.remove();
317 }
318
319 long endTime=System.currentTimeMillis();//fin de ejecucion
320 long totalTime=endTime-startTime;
321 System.out.println("Tiempo de ejecucion: "+totalTime+" milisegundos");
322 return costo;
323 }
324
325
326 public static void main(String[] args) {
327     MST a=new MST();
328     System.out.println(a.prim("P2Edges.txt"));
329     // System.out.println(a.kruskalDFS("arbol.txt"));
330     // System.out.println(a.kruskalDFS("P2Edges.txt"));
331     // System.out.println(a.kruskalUF("arbol.txt"));
332     // System.out.println(a.kruskalUF("P2Edges.txt"));
333 }
```

Console Output:

```
<terminated> MST (1) [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Apr 23, 2018, 11:26:05 PM)
(225 189 -2315)
(277 326 -2266)
(263 262 -2202)
(232 331 -1972)
(292 78 -1847)
(397 396 -1734)
(55 54 -1572)
(247 119 -1467)
(141 429 -1127)
(253 306 -952)
(313 487 -320)
(122 5 722)
(352 21 1462)
(198 23 3114)
Tiempo de ejecucion: 116 milisegundos
-3612829.0
```

#### 4. Función float kruskalDFS(String archivo)

Recibe el archivo y por medio del algoritmo de Kruskal implementado con búsqueda en profundidad nos permite encontrar el árbol de expansión mínima.



The screenshot shows the Eclipse IDE with the same 'proyecto2ada' project. The file 'MST.java' is open. The code is similar to the previous one, but the main method now calls the 'kruskalDFS' method instead of 'kruskalUF'. The console output shows the execution time and the total cost of the MST.

```
312         costo+=pq.peek().w;
313         pq.peek().imprime();
314         mst.add(pq.peek());
315     }
316     pq.remove();
317 }
318
319 long endTime=System.currentTimeMillis();//fin de ejecucion
320 long totalTime=endTime-startTime;
321 System.out.println("Tiempo de ejecucion: "+totalTime+" milisegundos");
322 return costo;
323 }
324
325
326 public static void main(String[] args) {
327     MST a=new MST();
328     //System.out.println(a.prim("P2Edges.txt"));
329     System.out.println(a.kruskalDFS("P2Edges.txt"));
330     // System.out.println(a.kruskalUF("arbol.txt"));
331     // System.out.println(a.kruskalUF("P2Edges.txt"));
332 }
333 }
```

Console Output:

```
<terminated> MST (1) [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Apr 23, 2018, 11:27:49 PM)
(277 326 -2266)
(262 263 -2202)
(232 331 -1972)
(78 292 -1847)
(396 397 -1734)
(54 55 -1572)
(119 247 -1467)
(141 429 -1127)
(253 306 -952)
(313 487 -320)
(5 122 722)
(21 352 1462)
(23 198 3114)
Tiempo de ejecucion: 250 milisegundos
-3612829.0
```

## 5. Función float kruskalUF(String archivo)

Recibe el archivo y por medio del algoritmo de Kruskal implementado con Union-Find, encuentra el árbol de expansión mínima, regresa lo siguiente:

```
326 public static void main(String[] args) {
327     MST a=new MST();
328     //System.out.println(a.prim("P2Edges.txt"));
329     //System.out.println(a.kruskalDFS("P2Edges.txt"));
330     System.out.println(a.kruskalUF("P2Edges.txt"));
331 }
332 }
333
```

Problems Javadoc Declaration Console Coverage

<terminated> MST (1) [Java Application] C:\Program Files\Java\jre1.8.0\_161\bin\javaw.exe (Apr 23, 2018, 11:29:07 PM)

(277 326 -2266)  
(262 263 -2202)  
(232 331 -1972)  
(78 292 -1847)  
(396 397 -1734)  
(54 55 -1572)  
(119 247 -1467)  
(141 429 -1127)  
(253 306 -952)  
(313 487 -320)  
(5 122 722)  
(21 352 1462)  
(23 198 3114)  
Tiempo de ejecucion: 120 milisegundos  
-3612829.0

## 6. Ejemplo de implementación:

Archivo de texto.

```
arbol.txt - N
File Edit Form
5 8
1 2 6
1 5 1
3 5 3
2 5 3
5 4 4
4 3 1
2 3 6
1 3 4
```

Usando Prim:

```
325
326 public static void main(String[]
327     MST a=new MST();
328     System.out.println(a.prim("ar
329     //System.out.println(a.kruska
330     //System.out.println(a.kruska
331 }
332 }
333
```

Problems Javadoc Declaration Console

<terminated> MST (1) [Java Application] C:\Program Files

Las aristas son:  
(1 5 1)  
(5 3 3)  
(3 4 1)  
(5 2 3)  
Tiempo de ejecucion: 31 milisegundos  
8.0

### Usando Kruskal con DFS:

```
325
326 public static void main(String[] args) {
327     MST a=new MST();
328     //System.out.println(a.prim("a
329     System.out.println(a.kruskalDFS("a
330     //System.out.println(a.kruska
331 }
332 }
333
```

<terminated> MST (1) [Java Application] C:\Program Files\Java\jre6\bin\java.exe  
Las aristas son:  
(1 5 1)  
(4 3 1)  
(2 5 3)  
(3 5 3)  
Tiempo de ejecucion: 63 milisegundos  
8.0

### Usando Kruskal con Union-Find

```
325
326 public static void main(String[] args) {
327     MST a=new MST();
328     //System.out.println(a.prim("arbol.t
329     //System.out.println(a.kruskalDFS("a
330     System.out.println(a.kruskalUF("arbo
331 }
332 }
333
```

<terminated> MST (1) [Java Application] C:\Program Files\Java\jre6\bin\java.exe  
Las aristas son:  
(1 5 1)  
(4 3 1)  
(2 5 3)  
(3 5 3)  
Tiempo de ejecucion: 33 milisegundos  
8.0