Adrián González Galván del Grupo 2 de Tarde

Actividad II: Árboles generadores de mínimo coste, el algoritmo de Kruskal (explicación sobre mí implementación en C++)

Antes de todo, recordemos que dentro de la función en la que implementamos el **Algoritmo de Kruskal** tenemos los siguiente recursos:

- Un **vector** denominado **Aristas** formado por elementos del tipo **AristaPesada** (estructura formada por 2 nodos (extremo1 y extremo2) y un peso) que contiene todas las aristas del grafo en cuestión.
- Una **variable** del **tipo unsigned** llamada **head** inicializada a cero (que nos sirve para inicializar la cabeza del vector anterior).
- Otra **variable** del **tipo unsigned** llamada **a** inicializada a cero (para contar el número de aristas que introducimos en la solución).
- Otra **variable unsigned** más llamda **pesoMST** que indica el peso de nuestro árbol generado como solución.
- Y un **vector** del tipo **unsigned** denominado **Raiz** para ir reconociendo las distintas componentes conexas de los nodos.

Así pues, mi **manera de implementar el algoritmo** fue la siguiente:

- Nos aseguramos antes de todo (con un if) que el **vector Aristas no esté vacío**, de no ser así el algoritmo terminará en el acto (dado a que de un grafo NO conexo no podemos obtener un árbol generador).
- Creamos **una variable "mincoste"** y guardamos en ella el peso de la arista a la que apunta la cabeza (head).
- Recorremos con un **bucle** el vector de **Aristas** desde la posición a la que apunta la cabeza hasta el final, buscando la **arista con menor coste**.
- Una vez encontrada dicha arista, **la permutamos**, es decir, la intercambiamos por la que se encuentra en la posición a la que apunta la cabeza (apoyándonos de una AristaDummy).
- ¡ Ya tenemos en la posición **Aristas[head]** la arista con menor coste a mostrar !
- **Revisamos** entonces (con un if) que los **extremos de estas aristas se encuentren en componentes conexas distintas**. Si es así, actualizamos y establecemos para cada nodo con Raiz[q] = raiz[extremo1] una nueva raíz: raiz[extremo2]. Del contrario, no mostramos dicha arista ya que con ella formaríamos un ciclo.
- **Si hemos mostrado dicha arista:** incrementaremos la variable peso con el peso de dicha arista y actualizaremos (sumaremos 1) el contador de aristas "a".
- Por otro lado, independientemente de que hayamos mostrado o no dicha arista, **incrementamos el valor de "head"** para que el nuevo bucle sitúe la siguiente arista más barata en la siguiente posición del vector.
- Una vez **acabado el bucle do-while** (cuando hayamos mostrado a==(n-1) aristas y el head haya recorrido todas las posiciones del vector) se habrán mostrado satisfactoriamente todas las aristas de nuestra solución.

No tuve grandes dificultades con la realización de esta práctica. Casualmente, donde tuve el mayor problema fue con el **uso del vector raíz** (el cual no lo implementé eficientemente las primeras veces porque me generaba ciclos en la solución). Pero resultó ser nada más que **una mala inicialización del entero "kill"**.
Por otro lado, en la **ordenación de las aristas de menor a mayor coste** no tuve ningún problema. Incluso me aseguré de mostrarlas por pantalla a medida que las iba ordenando para ver si estaba correctamente implementado y así fue.