EL ALGORITMO DE KRUSKAL

Memoria de la práctica

Autor: Jaime Pastrana García

Introducción

Mediante esta memoria se pretende documentar el desarrollo y los resultados de la práctica. Para ello, se describe brevemente la implementación realizada para el algoritmo de Kruskal, además de indicarse la manera de ejecutar el código con los ficheros de entrada. A continuación, se explican los costes teóricos del algoritmo dadas las estructuras de datos empleadas y, finalmente, dichos costes se comparan mediante gráficas con los resultados obtenidos en tiempos de ejecución reales.

Explicación breve de la implementación

En primer lugar, para poder representar los grafos, se usan listas de adyacencia. Se prefiere a una matriz por motivos de eficiencia. Además, se emplea una estructura de partición para poder llevar el subconjunto al que pertenece cada arista de manera eficiente. Todo esto se implementa mediante clases. Además, se emplea un struct para las aristas con el objetivo de facilitar el manejo de estas en el algoritmo de Kruskal.

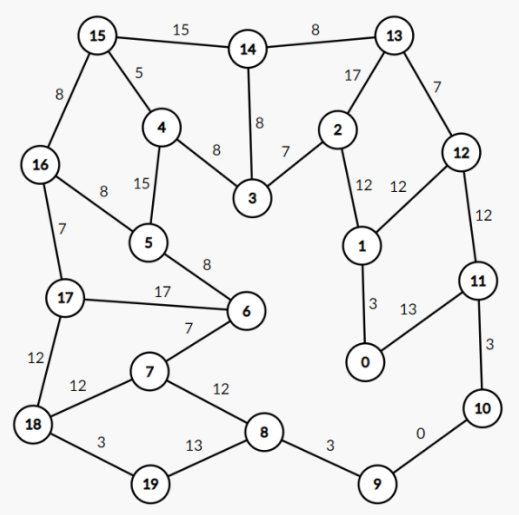
Dadas las estructuras de datos, resta razonar el algoritmo de Kruskal en sí mismo. Para ello, tal y como se ha explicado en la asignatura, se procede en los siguientes pasos:

1. Inicializar un vector de aristas vacío (vector del ARM[1]) y un vector que contenga las aristas del grafo (ordenadas por pesos). También inicializar una estructura de partición que guarde correspondencia de los vértices (cada vértice en un subconjunto independiente inicialmente).
2. Se recorren las aristas en orden creciente, analizando si los vértices que conecta pertenecen a distintos subconjuntos. Si no lo hace es porque forma un ciclo, por lo que se descarta. Si efectivamente pertenecen a distintos subconjuntos, se fusionan ambos subconjuntos en la estructura de partición y se añade la arista al vector del ARM.
3. Se procede de esta manera hasta que se acaben las aristas o el ARM haya alcanzado tantas aristas como número de vértices menos uno. Esto es porque todo árbol tiene tantas aristas como nodos menos uno.

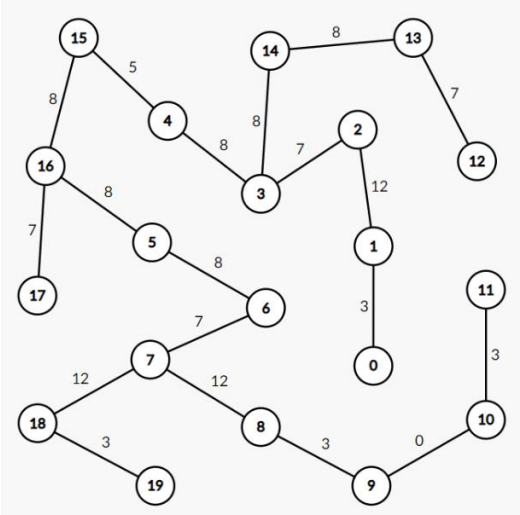
Resumiendo de manera metafórica el funcionamiento del algoritmo, se trata de tener originalmente tantas gotitas como vértices e ir juntándolas por donde estén más cerca hasta que estén todas conectadas en una gota grande. En ese momento se habrá obtenido una gota gigante conectada por las aristas de menor peso posible.

Ejemplo del resultado de ejecutar el algoritmo de Kruskal sobre un grafo de ejemplo:

[1] ARM: Árbol de recubrimiento de coste mínimo



Grafo original



Grafo del ARM resultante

Uso del programa

La compilación del programa es equivalente a la de cualquier otro archivo .cpp de c++. El texto fuente contiene toda la funcionalidad y no precisa de archivos de cabecera o similares.

Para ejecutar el programa no se precisa de añadir argumentos de entrada. Se ha procurado ofrecer una funcionalidad lo más intuitiva posible. De esta manera, se tiene un formato de menús en los que se introduce mediante un número la opción elegida. Se ha implementado la opción tanto de ejecutar el algoritmo sobre un grafo concreto como sobre un conjunto de grafos (esto último con la finalidad de obtener datos sobre los tiempos de ejecución en distintos tamaños). Para el correcto funcionamiento del programa, se deben colocar todos los ficheros que contienen los grafos en el mismo directorio que el programa ejecutable.

Los grafos generados para las mediciones de tiempo y pruebas del programa se encuentran en la carpeta “Ficheros de grafos”. Los ficheros que contienen los grafos no tienen extensión, aunque pueden ser abiertos como texto en formato UTF-8. La representación que emplean del grafo presenta número de vértices, número de aristas y una lista de todas la aristas como pares de vértices y peso asociado. De este modo, si se desea y el grafo no es muy grande, se puede visualizar en alguna de las múltiples páginas web especializadas en ello.

Explicación de los costes (costes teóricos y datos obtenidos en las gráficas)

En toda esta sección se presupondrá que a es el número de aristas y v el número de vértices de nuestro grafo. Bajo estas premisas, el coste del algoritmo de Kruskal bajo las estructuras de datos empleadas está en $O(a*\log v)$. Este coste se puede deducir del coste de cada uno de los pasos que anteriormente se han explicado. Sus respectivos costes serían:

1. El coste del primer paso de inicialización es $O(a*\log a)$. Esto es porque la operación más costosa es la de ordenar el vector de aristas.
2. En este paso, el coste amortizado de modificar la estructura de partición es $O(\alpha(n))$. Sin embargo este coste se puede aproximar a $O(1)$. Esto es porque α (la función de Ackermann) es una función que crece muy lentamente.

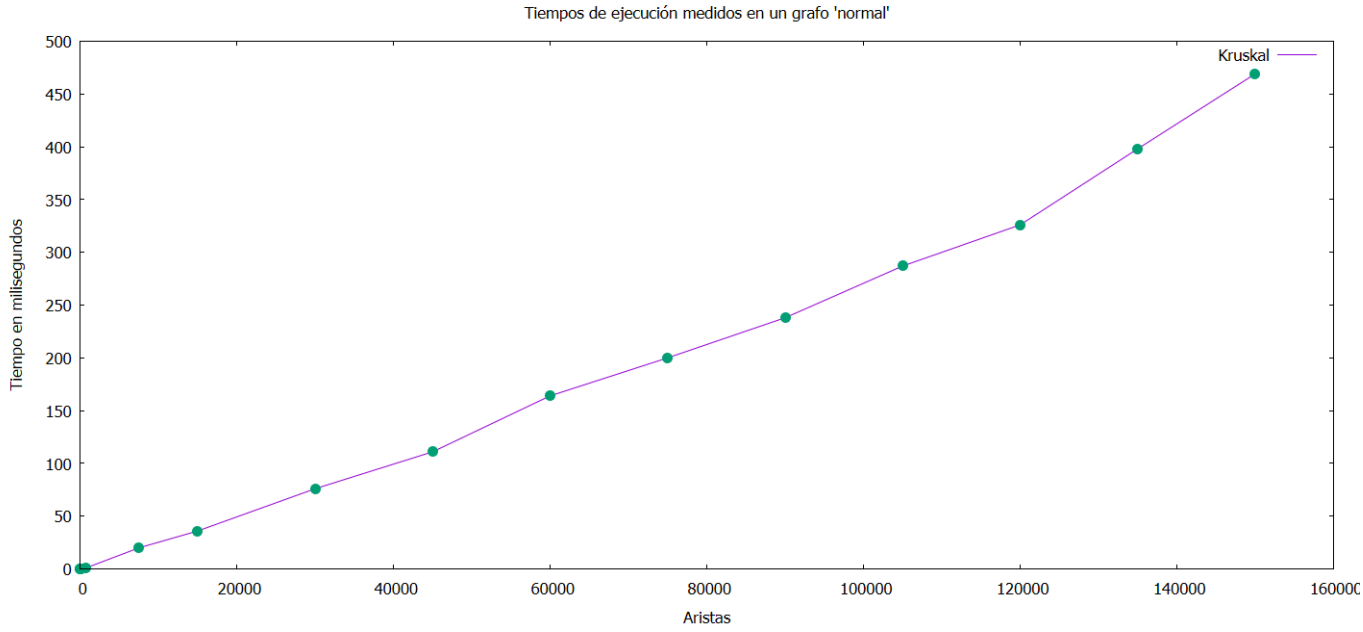
3. En este paso (que es la iteración del anterior), obtenemos un coste de $O(v)$, dado que como mucho recorremos $v-1$ aristas.

El coste final en el que se pueden englobar las operaciones de este algoritmo es del orden de $O(a*\log v)$. Esto es por dos motivos. El primero es que $O(a*\log a)$ es la operación más costosa del algoritmo. El segundo es que se puede razonar que $O(a*\log a)$ está en el mismo orden de coste que $O(a*\log v)$. Esto es porque, en el peor de los casos, $a = v^2$. Por ello se puede deducir que $O(a*\log a) = O(a*\log v^2) = O(a*2*\log v) \in O(a*\log v)$.

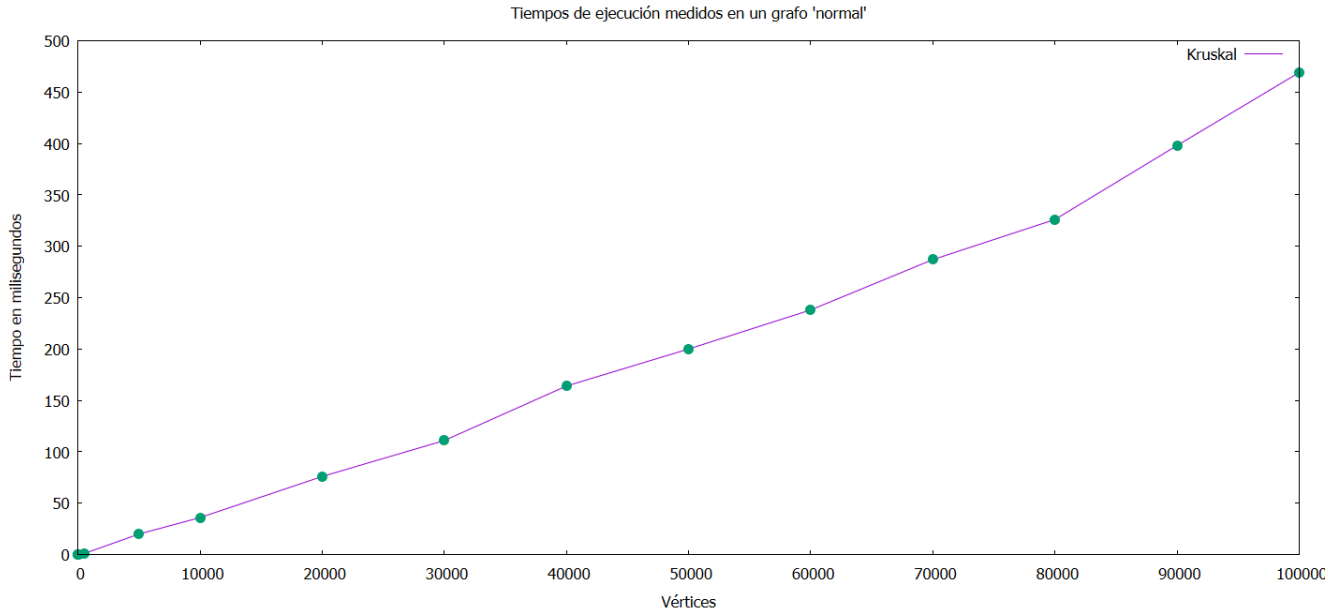
Llegados a este punto, sólo resta comparar los costes teóricos razonados con los costes reales obtenidos de la ejecución del algoritmo. Para ello se emplearán dos tipos de grafos: grafos "normales" (que mantienen una relación lineal entre el número de aristas y el número de vértices) y grafos completos. Además, para cada conjunto de grafos, se medirá el tiempo de ejecución en relación tanto a las aristas como los vértices.

1. Grafos "normales"

En este conjunto de grafos los vértices y aristas guardan una relación tal que a = 3/2*v.



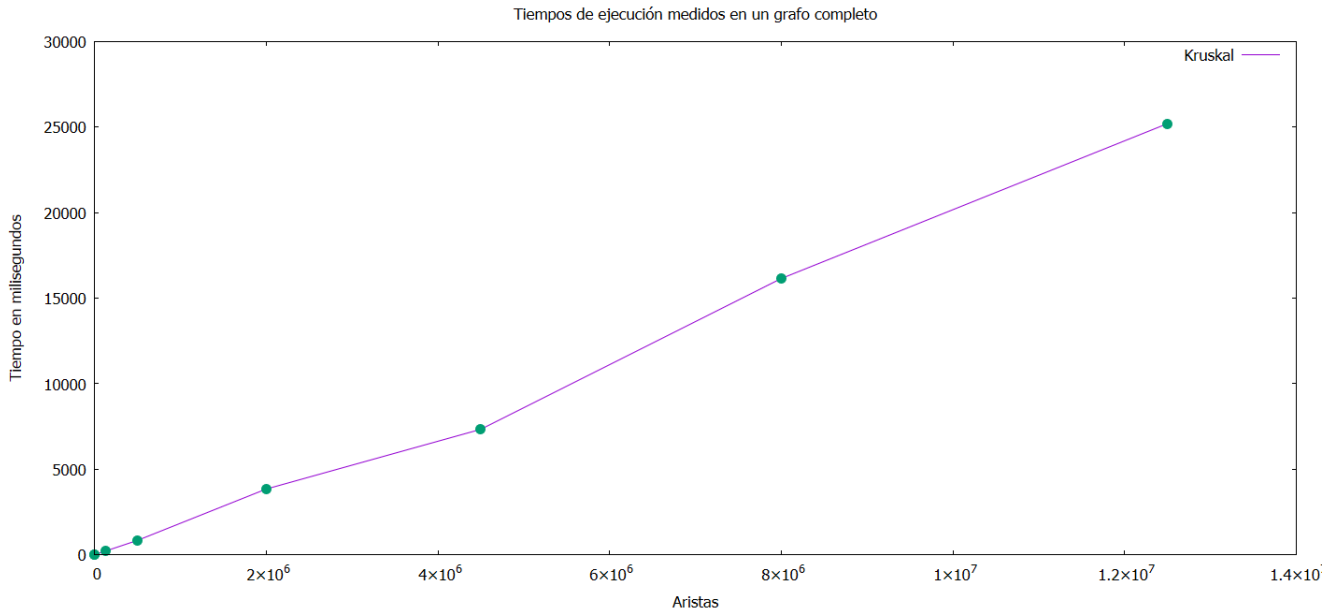
Como se puede observar, los resultados entran dentro de lo esperado. Se tiene, por tanto, una relación casi lineal entre las aristas del grafo y el tiempo de ejecución. Sin embargo, no es completamente lineal debido al producto logarítmico que se tiene asociado.



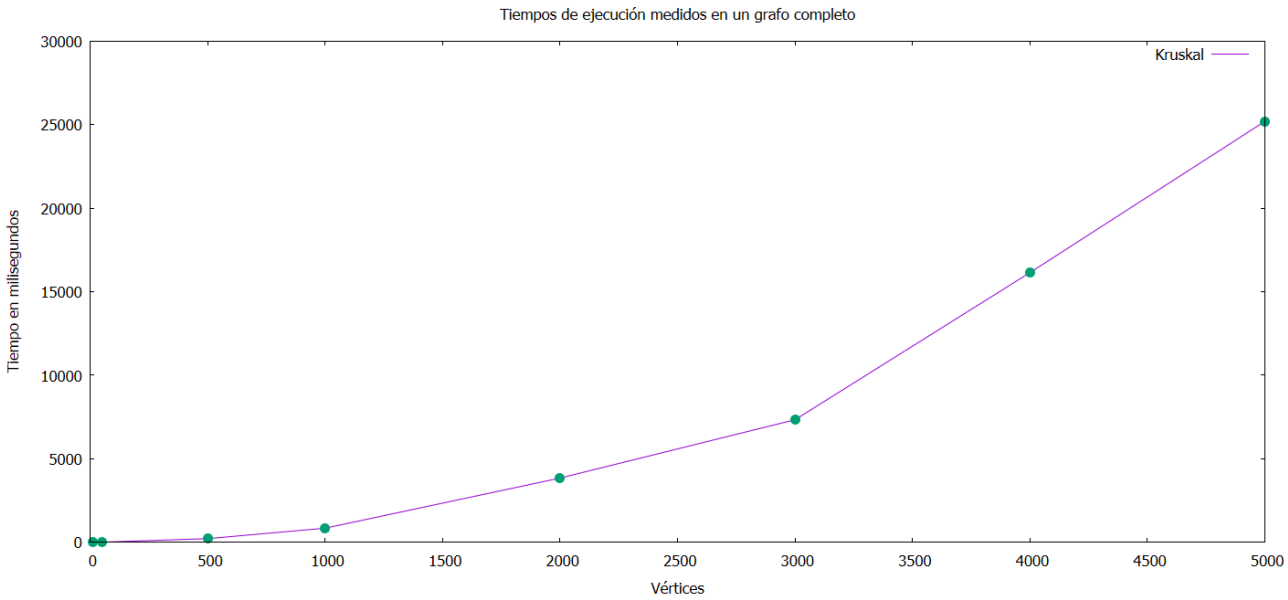
En este gráfico se obtiene un resultado similar al anterior dado que se mantiene una relación lineal entre las aristas y los vértices.

2. Grafos completos

En este caso las aristas serán el cuadrado de los vértices, al tratarse de grafos completos.



Se comprueba como se mantiene la relación casi lineal entre las aristas y el tiempo de ejecución. Esto es por el coste que tenemos en función de las aristas tal que $O(\mathbf{a}*\log a)$.



En este caso, en el que se relaciona el tiempo de ejecución con los vértices, se observa una gráfica exponencial. Esto se corresponde con lo esperado, dado que el coste en función de los vértices sería tal que $O(\mathbf{v^2}*\log v^2)$. De hecho, se puede observar como los tiempos de ejecución se disparan hasta los 26 segundos con tan sólo 5000 vértices. Esto es porque las aristas mantienen una relación cuadrática con los vértices.

Conclusión

A modo de conclusión, se puede afirmar que el algoritmo de Kruskal es un algoritmo eficiente cuando el número de aristas de un grafo es relativamente bajo. Constituye, por tanto, un buen mecanismo para encontrar ARMs en grafos de muchos vértices pero una baja proporción de aristas. En cualquier caso, hay que tener en cuenta que, si no reunimos dichas condiciones, el algoritmo puede resultar tremendamente ineficiente. Esto se puede observar claramente al comparar los resultados obtenidos con 5000 vértices para un grafo completo respecto a un grafo normal. Mientras que en un grafo completo el tiempo de ejecución medido se encuentra entorno a los 25 segundos, en un grafo "normal" se queda en menos de un cuarto de segundo.

Bibliografía

- Para la representación de los datos de tiempo de ejecución en gráficas se ha usado gnuplot.
- Documento original en el que el autor original del algoritmo explica se funcionamiento (1956): https://www.ams.org/journals/proc/1956-007-01/S0002-9939-1956-0078686-7/