Introducción

Los algoritmos Greedy son los también conocidos como algoritmos voraces, entre los que encontramos los que se han propuesto para la implementación de esta práctica.
Estos algoritmos se basan en una estrategia de búsqueda basada en una heurística que escoge la opción óptima de manera local con la intención de lograr una solución óptima para el problema completo.
Por ello, y basándonos en lo estudiado en la asignatura se han implementado distintos algoritmos en base al pseudocódigo proporcionado por los profesores y adaptados para poder usarlos con los datos proporcionados en la asignatura de EDAI.
El primero de los algoritmos que se han utilizado es Prim, algoritmo que busca encontrar un árbol recubridor mínimo en un grafo conexo, no dirigido y cuyas aristas están etiquetadas.
El segundo algoritmo es Kruskal, algoritmo que busca encontrar un árbol recubridor mínimo en un grafo conexo y ponderado.

Estudio teórico

Para el estudio teórico de esta práctica comenzaremos analizando los órdenes de complejidad teóricos de los distintos algoritmos que vamos a usar.
Comenzando por el algoritmo de Prim que posee un orden de complejidad $O(n^2)$ en el peor de los casos y O(m log n) en el caso de usar colas de prioridad siendo n = número de nodos del grafo y m = número de aristas.
Kruskal realmente posee los mismos órdenes de complejidad, entonces..
¿Cuando es mejor usar uno y cuando otro?
La respuesta a esta pregunta radica en la complejidad del grafo en un ARCM dado que por la naturaleza del algoritmo de kruskal depende más del número de aristas, si el grafo es muy denso es probable que el orden de complejidad acabe siendo similar a $O(n^2 \log n)$, es en estos casos cuando Prim es más eficiente, sin embargo cuando el grafo es más disperso (posee menos aristas) Kruskal es preferible.
Siguiendo el analisis teorico podemos observar como en nuestro caso poseemos un grafo con una densidad algo elevada y por lo tanto es mejor usar el algoritmo de Prim, la diferencia es cada vez más notable a medida que aumentamos los nodos, este aumento muestra también como en casos con grafos pequeños el algoritmo de kruskal no se beneficia de una cola de prioridad si no que más bien esto es contraproducente, sin embargo cuando el número de nodos llega, en nuestro caso, a 625, la cola de prioridad hace que el tiempo disminuye en aproximadamente un 50%.

Cuestiones teóricas:
1-¿El resultado de la ejecución de cada algoritmo es único?
Si, la forma de trabajar de cada algoritmo es diferente lo cual lleva a diferentes resultados estando por ejemplo el resultado de Prim en un orden y el de Kruskal en otro debido a la ordenación.

2- ¿El resultado de la ejecución de los dos algoritmos debe ser el mismo?, ¿por qué?.

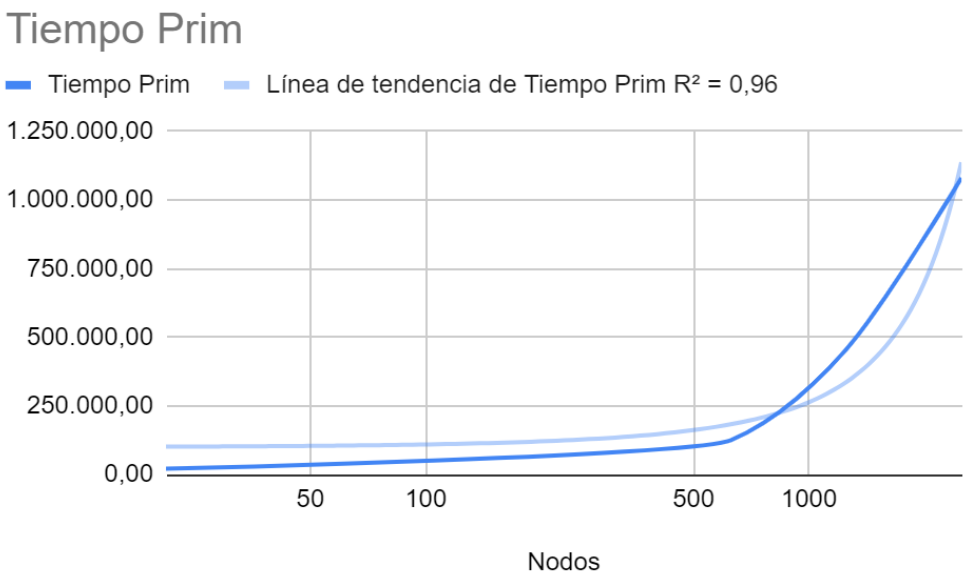
No, puede serlo pero no obligatoriamente van a ser iguales ya que la forma de trabajar de cada algoritmo lleva a resultados diferentes pero que pueden coincidir.

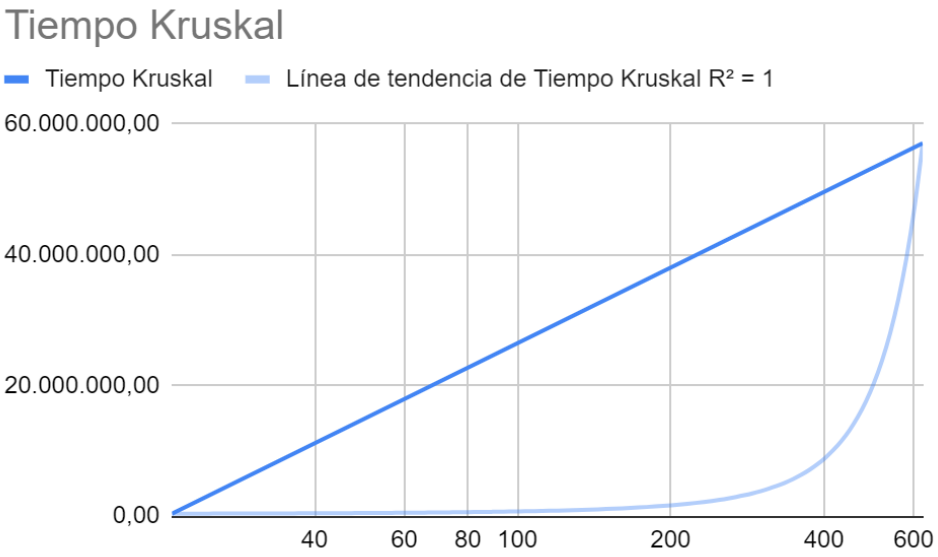
3-Si el peso de las aristas fuese la distancia entre dos ciudades, con la estructura resultante, ¿podemos determinar el camino mínimo entre dos pares de ciudades cualquiera?.
No, puede que el camino que encontremos sea el mínimo pero no tiene por qué ser la óptima debido a la forma que trabajan los algoritmos Greedy

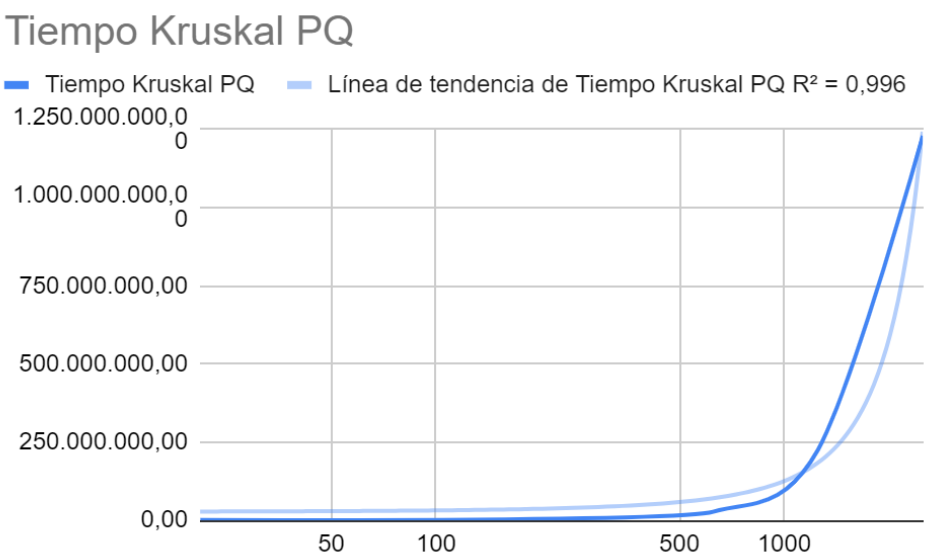
Estudio experimental

A continuación se exponen los resultados obtenidos en nuestro código con el archivo EDALand, y las redes generadas con 625, 1250 y 2500 nodos. No hemos podido realizar ciertas tomas de datos debido a la limitada capacidad de nuestros ordenadores, la cual hacía que superado cierto número de nodos la cantidad de tiempo que debíamos esperar era especialmente alta.

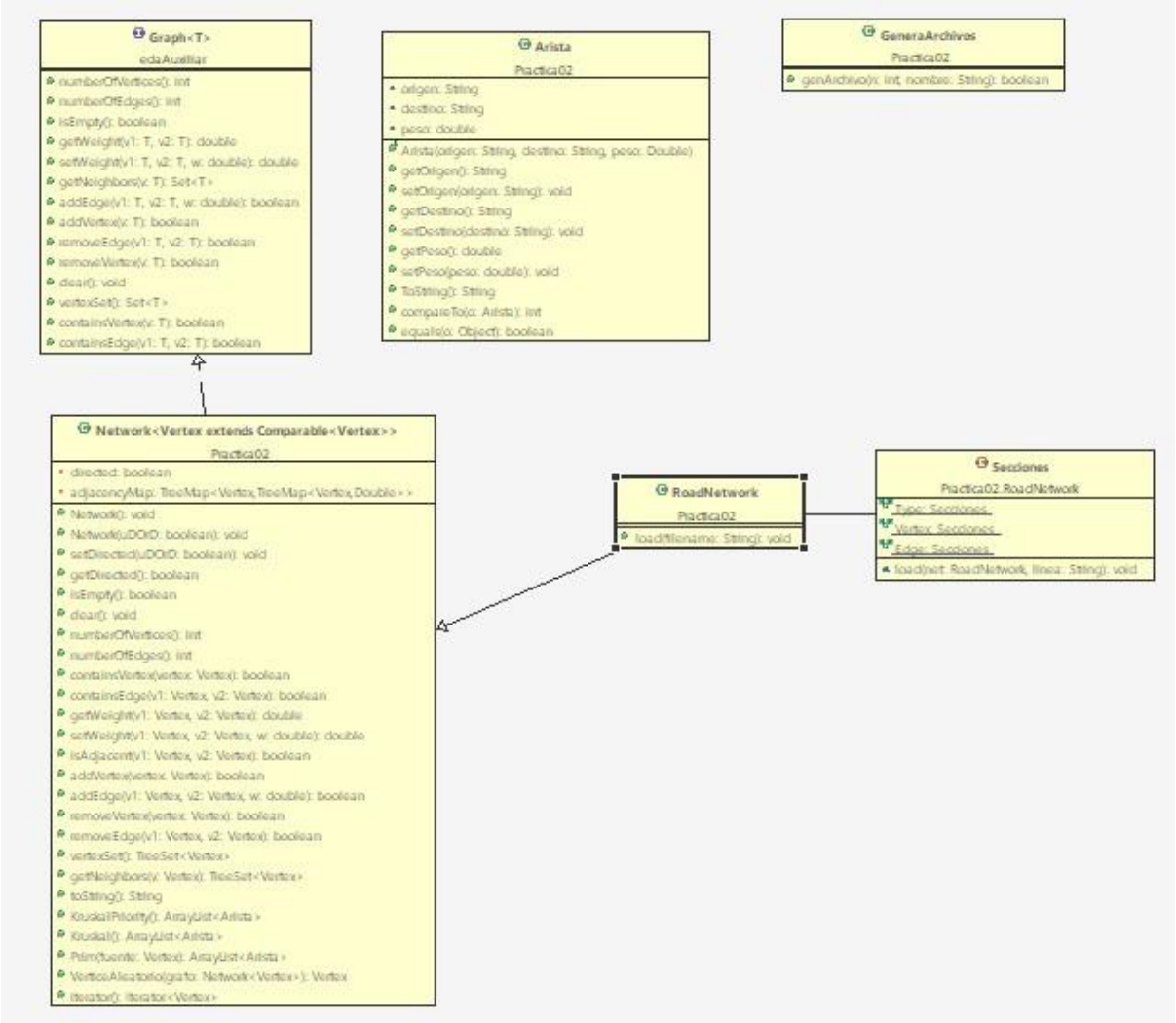
Nodos	Tiempo Prim	Tiempo Kruskal	Tiempo Kruskal PQ
21(EdaLand)	21.155,00	371.166,00	882.288,00
625	124.766,67	57.050.000,00	27.390.000,00
1250	455.200,00		225.000.000,00
2500	1.080.588,89		1.230.000.000,00




Tiempo Kruskal
Tiempo Kruskal
Línea de tendencia de Tiempo Kruskal R² = 1
60.000.000,00
40.000.000,00
20.000.000,00
0,00
40
60
80
100
200
400
600


Tiempo Kruskal PQ
Tiempo Kruskal PQ
Línea de tendencia de Tiempo Kruskal PQ R² = 0,996
1.250.000.000,00
1.000.000.000,00
750.000.000,00
500.000.000,00
250.000.000,00
0,00
50
100
500
1000

ANEXO



RoadNetwork-> tratamiento del archivo fuente.
Arista -> como su propio nombre indica se trata de la clase Arista y métodos necesarios para su tratamiento.
GeneraArchivos -> generación de archivos aleatorios a partir de un n dado.
Generar_archivo -> usado para generar archivos de distintos tamaños.
Network-> implementación de la red a estudiar por los algoritmos.
Prim_Kruskal_Aleatorio-> distintas pruebas para la comprobación de los algoritmos.
Prim_Kruskal_Clase-> main para estudio experimental.
Prim_Kruskal_Edalang-> pruebas de comprobación de los algoritmos con la entrada del archivo EDAland.
Prim_Kruskal_EdalangLarge-> pruebas de comprobación de los algoritmos con la entrada del archivo EDAlandLarge.

Bibliografía

- PDF de la asignatura
- https://www.wextensible.com/temas/voraces/kruskal-prim.html