GUION DE LA PRÁCTICA 4 (directorio p4)

0)INTRODUCCIÓN

El algoritmo de Prim es uno de los ejemplos más claros (tal vez el mejor) de algoritmo devorador o ávido. A partir de un grafo de n nodos, va eligiendo de forma segura y rápida, arista a arista, hasta seleccionar n-1 aristas, obteniendo así la forma de tener conectados los n nodos con menor coste (coste óptimo).

Como siempre, a la hora de implementar un algoritmo, las estructuras de datos que se elijan determinarán una complejidad mayor o menor. Para la implementación que se nos pide posteriormente de este algoritmo, se aceptará que tenga cualquier complejidad, si bien se valorará más mientras menor sea la complejidad.

1)ALGORITMO DE PRIM

Se le proporciona el módulo **Auxiliar.py** ,que tiene unas funciones que permiten crear en una matriz un grafo completo de aristas bidireccionales (con pesos aleatorios en el intervalo [100..999]. Por supuesto, puede modificar y añadir en ese módulo todo lo que considere apropiado.

Los ficheros **grafo4.txt, grafo8.txt, … grafo256.txt** contienen ejemplos de grafos, su formato supone colocar en la primera línea el número n de nodos, en la segunda línea los pesos de las aristas del primer nodo (nodo 0) a todos los demás (por orden), en la tercera línea los pesos de las aristas del segundo nodo (nodo 1) a todos los demás de mayor índice, y así sucesivamente.

Los ficheros **resultado4.txt, resultado8.txt, … resultado256.txt** contienen la solución, calculada por Prim, para cada uno de los grafos de entrada vistos antes. En el caso de que hubiere varias soluciones óptimas con el mismo coste mínimo nos vale cualquiera de ellas, por lo que podría haber otras soluciones válidas a las puestas, pero evidentemente han de tener el mismo coste mínimo óptimo.

SE LE PIDE:

Implementar un módulo Prim.py, de forma que tras meter el nombre de un fichero, calcule y escriba por pantalla la solución óptima.

Razone convenientemente la complejidad temporal del algoritmo implementado.

Implementar un módulo PrimTiempos.py, que vaya creando grafos aleatorios de diversos tamaños y calculando el tiempo que tarda el algoritmo hecho en el apartado anterior en resolver el problema, de forma que se pueda ir rellenando la siguiente tabla.

Razone: ¿siguen la complejidad calculada los tiempos de la tabla?

TABLA TIEMPOS PYTHON ALGORITMO PRIM

(tiempos en milisegundos):

Pondremos "FdT" para tiempos superiores a los 10 minutos.

n	*t Prim*
256	
512	
1024	
2048	
4096	
8192	
16384	
… (hasta FdT)	

Se ha de entregar en un **.pdf** el trabajo que se le pide y además los módulos **.py** que ha programado. Todo ello lo organizará en una carpeta, que es la que entregará comprimida en un fichero **practica4ApellidosNombre.zip**.

Esta práctica es para una semana y el **plazo límite** de entrega de esta práctica es **un día antes** de que comience la siguiente práctica, en la tarea que se creará a tal efecto en el Campus Virtual.