Diseño y Análisis de Algoritmos

Práctica 2

Árboles abarcadores mínimos:
Multigrafos. Puntos de articulación. Algoritmo de Kruskal.

Andrián Navas Ajenjo
Gloria del Valle Cano
adrian.navas@estudiante.uam.es & gloria.valle@estudiante.uam.es

Grado en Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
18-10-2019

Índice general

Índice de figuras

1. Introducción

En la segunda práctica hemos implementado algoritmos básicos para multigrafos, detección de puntos de articulación, TAD conjunto disjunto y Kruskal.
Asimismo, abordaremos los problemas propuestos con la representación de gráficas que nos ayuden a realizar un análisis de los tiempos de los algoritmos con el fin de abstraer la complejidad y la estrategia de los algoritmos.

2. Cuestiones

Cuestiones sobre puntos de articulación

Cuestión 1

Tomando como base el código de las funciones `o_a_tables` y `p_o_a_driver`, dar razonadamente una estimación teórica del coste de detectar si un grafo conexo tiene o no puntos de articulación. Contrastar este análisis con las gráficas a elaborar mediante la función `time_pda` considerando únicamente grafos con `prob` 0.7 y 0.9.

```
times = time_pda(100,10,100,5,0.8)
plt.plot(list(times.keys()),list(times.values()))
plt.show()
```

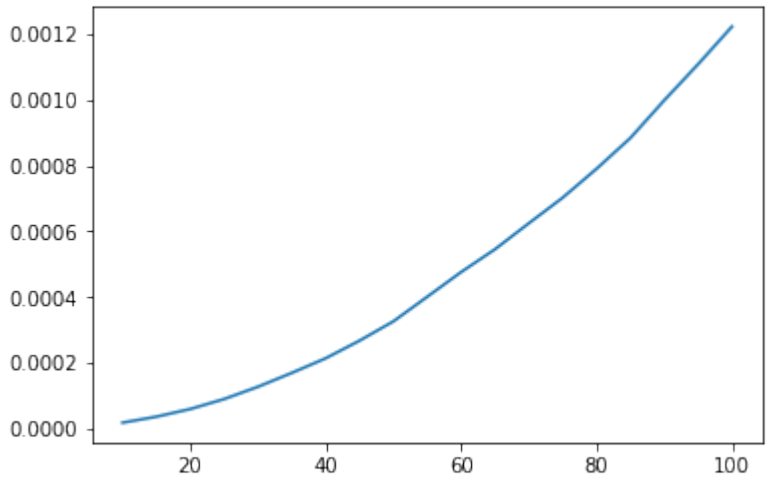

Figura 2.1: Comparación de tiempos de Kruskal2

Cuestión 2

¿Tiene sentido el concepto de punto de articulación en multigrafos? Si crees que sí, arguméntalo. ¿Cómo los definirías?¿Y qué habría que cambiar en las funciones anteriores para que funcionen en multigrafos?

Cuestiones sobre Kruskal

Cuestión 1

Discutir la aportación al coste teórico del algoritmo de Kruskal tanto de la gestión de la cola de prioridad como la del conjunto disjunto. Intentar llegar a la determinación individual de cada aportación.:

- **Cola de prioridad**:
  - Para insertar elementos en la cola de prioridad: O(|V|log|V|).
  - Para extraer elementos en la cola de prioridad: O(log|V|).
  - Finalmente el coste total de la cola de prioridad es de O(|V|log|V|).
- **Conjunto disjunto**:
  - Para inicializar el conjunto disjunto: O(|V|).
  - El coste de `find` acumulado es de O(|E|log*|V|), lo que es en definitiva O(|E|).
  - El coste acumulado de `union` es de O(|V|)
- **Coste total**: Debido a la aportación de cada operación se estima que al menos el coste de Kruskal total debe ser de:

$$O(|E| \log_{|} V|) \tag{2.1}$$

Cuestión 2

Contrastar la discusión anterior con las gráficas a elaborar mediante las funciones desarrolladas en la práctica.

```
times = time_kruskal(50,10,100,5,0.5,False)
plt.plot(list(times.keys()),list(times.values()))
plt.show()
```

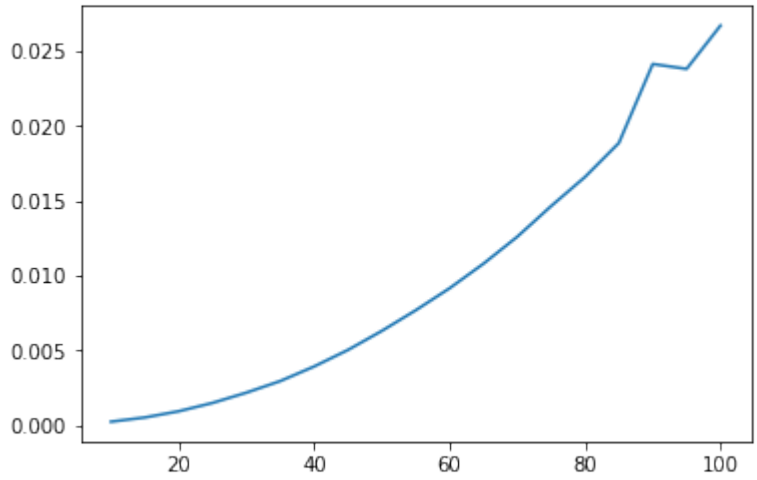

Figura 2.2: Comparación de tiempos de Kruskal

```
times = time_kruskal_2(50,10,100,5,0.5,False)
plt.plot(list(times.keys()),list(times.values()))
plt.show()
```

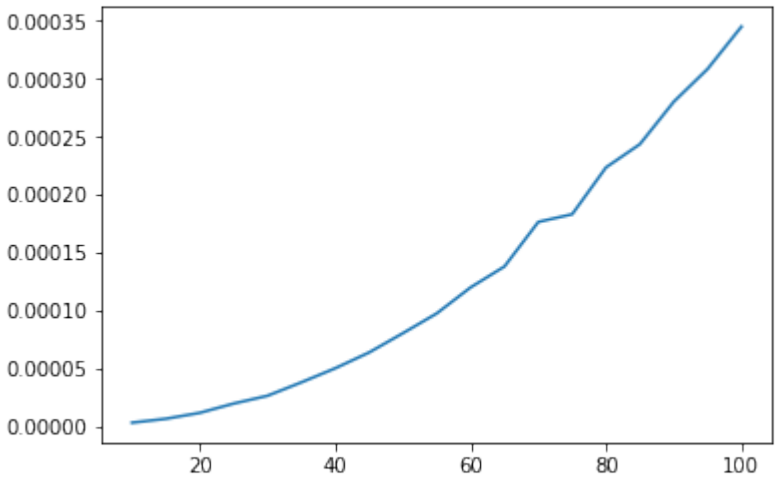

Figura 2.3: Comparación de tiempos de Kruskal2

Cuestión 3

¿Tiene sentido el concepto árbol abarcador mínimo en multigrafos? Si crees que sí, ¿cómo los definirías?¿Y qué habría que cambiar en las funciones anteriores para que funcionen en multigrafos?