Tarea 3

5272

1. Algoritmos para grafos

1.1. Todos los caminos más cortos

Se utiliza en grafos ponderados dirigidos conectados $G(V, E)$, donde para cada borde $< u, v > \in E$ se le asocia un peso w para el problema de todos los pares de rutas más cortas. Tiene aplicación en los problemas del servicio urbano, como la ubicación de las instalaciones urbanas o la distribución o entrega de bienes [5].

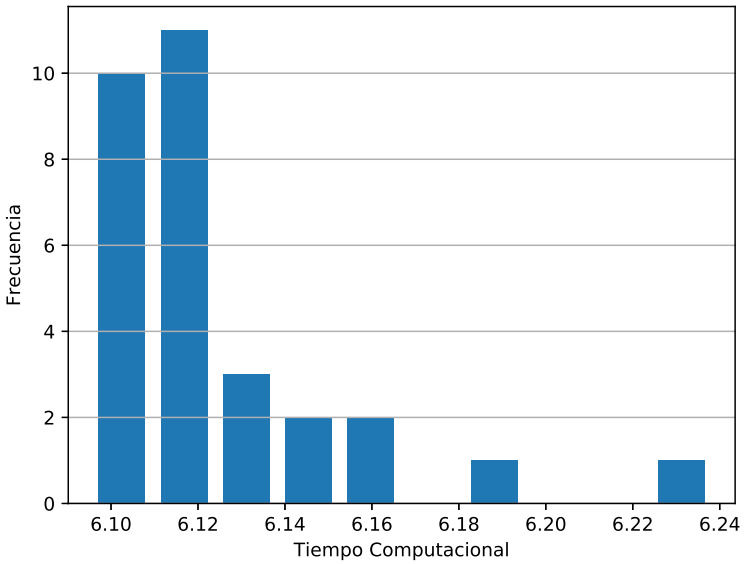

Figura 1

1.2. Centralidad intermedia

Este algoritmo calcula la ruta más corta ponderada entre cada par de nodos en un grafo conectado, en este se emplea la búsqueda por amplitud.Se utiliza para encontrar nodos que sirven de puente de una parte de un grafo a otra. Se aplica en un proceso de entrega de paquetes o red de telecomunicaciones [4].

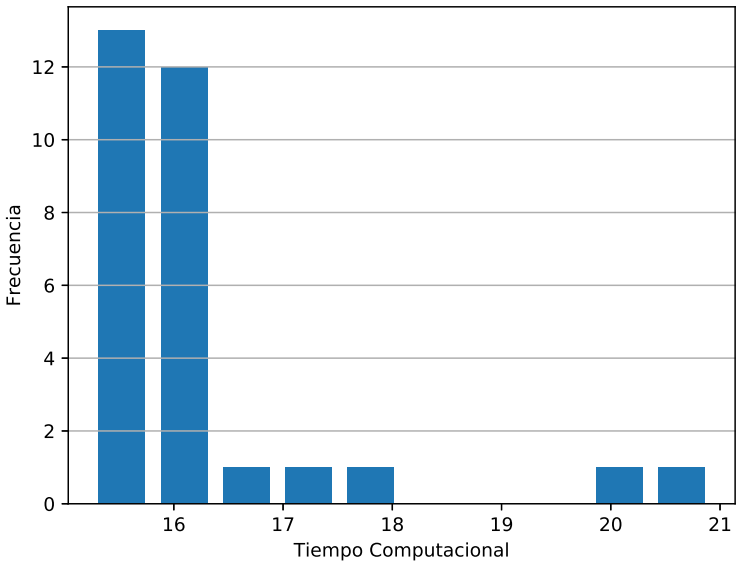

Figura 2

1.3. Búsqueda topológica

Este algoritmo crea un ordenamiento lineal de los vértices, de modo que si aparece el borde (u, v) en el gráfico, v aparece antes que u en el ordenamiento. El grafo debe de ser un gráfico acíclico dirigido [3].

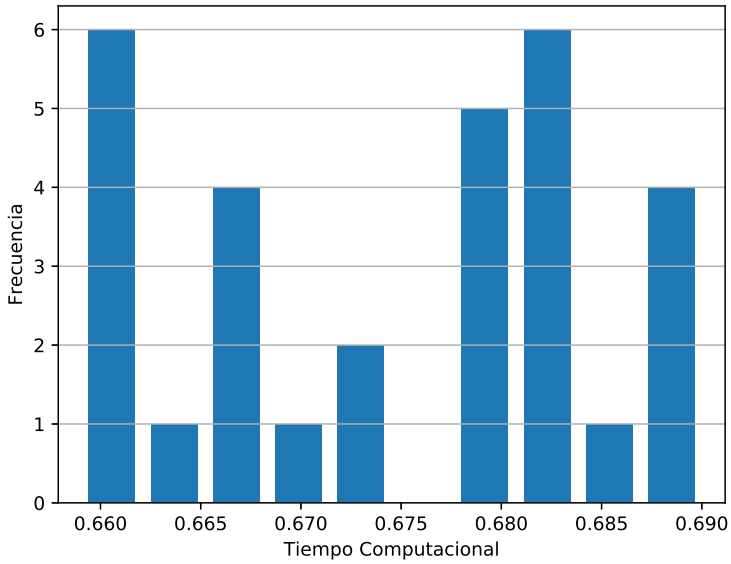

Figura 3

1.4. Camarilla máxima

Este algoritmo calcula la camarilla máxima del grafo G, es decir, el subgrafo completo del tamaño máximo. El parámetro *ind* es para la elección del método, si el parámetro es 0 el método es un algoritmo basado en la programación cuadrática $0 - 1$. El tamaño de la salida es el número de nodos de la camarilla encontrados por el algoritmo [2].

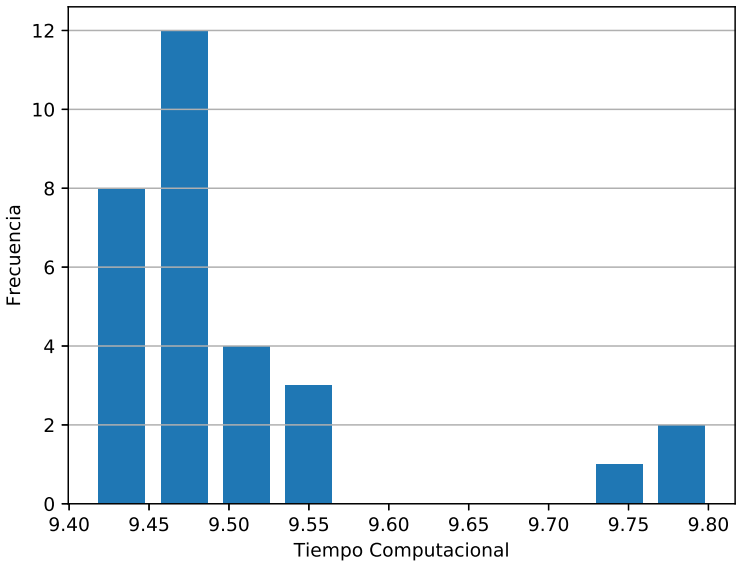

Figura 4

1.5. Árbol dfs

Este algoritmo de búsqueda de profundidad es una de las técnicas de desplazamiento gráfica más utilizadas; implica atravesar el grafo y construir un árbol de expansión (o bosque) arraigado [1].

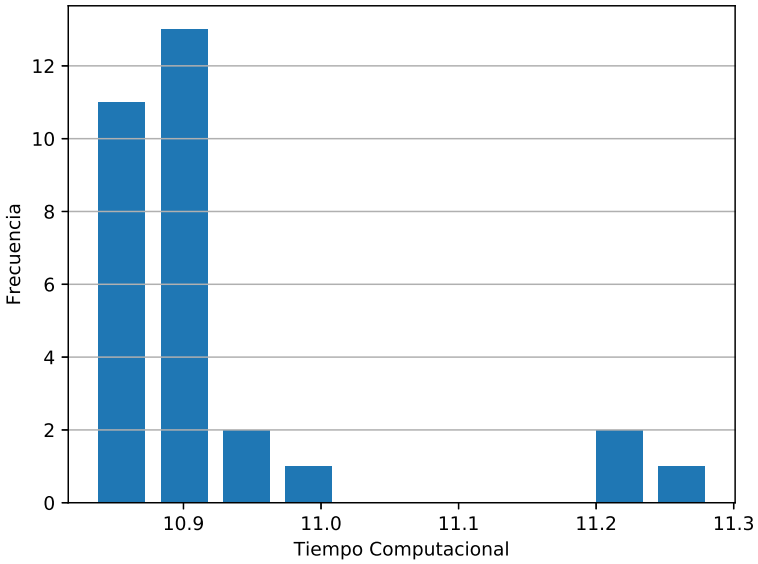

Figura 5

2. Análisis de resultados

Luego de correr los 5 algoritmos para grafos diferentes con un tiempo computacional mayor a 5 segundos con un número de replicas igual a 70 000 y realizar los histogramas correspondientes a cada uno, se demuestra que los datos no tienen una distribución normal.

Se realizan dos gráficos de dispersión Tiempo vs. Cantidad de nodos [Figura 6] y Tiempo vs. Cantidad de arista [Figura 7] donde se observa que el algoritmo más lento es el de Centralidad intermedia y el más rápido es Búsqueda topológica.

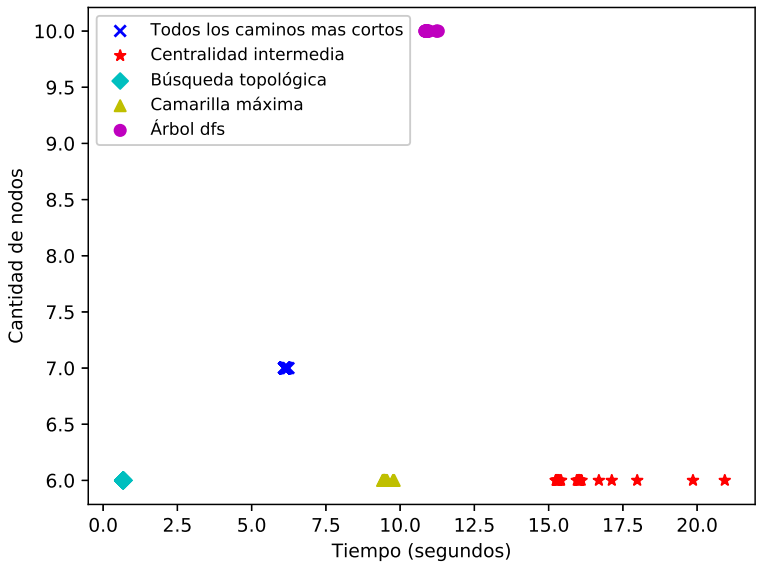

Figura 6: Tiempo vs Cantidad de Nodos

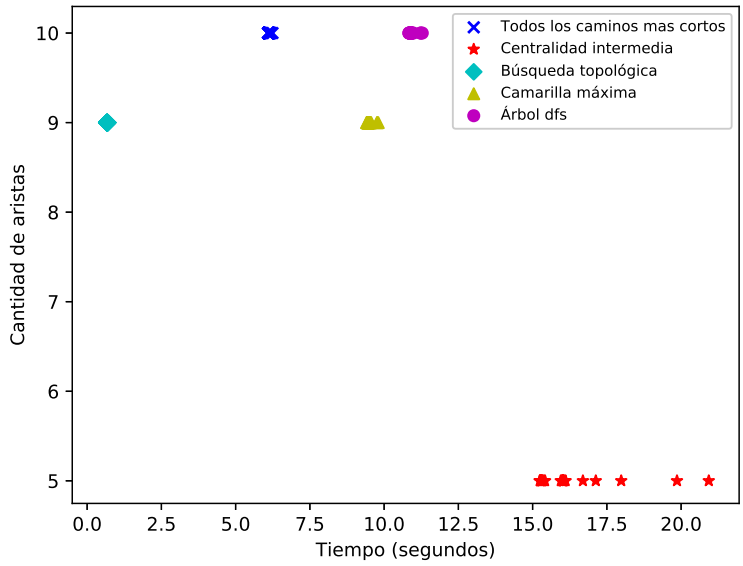

Figura 7: Tiempo vs Cantidad de Nodos

3. Fragmento de Código

```python
import networkx as nx
import matplotlib.pyplot as plt
import time
import numpy as np
import pandas as pd
from scipy import stats

#####################################################################
hist, bin_edges = np.histogram(list_5, density=True)
first_edge, last_edge = np.min(list_5), np.max(list_5)

n_equal_bins = 10
bin_edges = np.linspace(start=first_edge, stop=last_edge, num=n_equal_bins + 1, endpoint=True)

plt.hist(list_5, bins=bin_edges, rwidth=0.75)
plt.xlabel('Tiempo Computacional')
plt.ylabel('Frecuencia')
plt.grid(axis='y', alpha=0.75)
plt.savefig("Tarea3_05.eps")
plt.show(1)

normality_test = stats.shapiro(list_5)
print(normality_test)

#####################################################################

colors = ['b', 'c', 'y', 'm', 'r']

lo = plt.scatter(list_1, nodos_1, marker='x', color=colors[0])
ll = plt.scatter(list_2, nodos_2, marker='*', color=colors[4])
l  = plt.scatter(list_3, nodos_3, marker='D', color=colors[1])
a  = plt.scatter(list_4, nodos_4, marker='^', color=colors[2])
h  = plt.scatter(list_5, nodos_5, marker='o', color=colors[3])

plt.legend((lo, ll, l, a, h,),
           ('All-Pairs shortest paths', 'Betweenness Centrality', 'Topological Sort', 'Max Clique', 'DFS Tree'),
           scatterpoints=1,
           loc='upper left',
           ncol=1,
           fontsize=9)

plt.xlabel('Tiempo (segundos)')
plt.ylabel('Cantidad de nodos')
plt.savefig("Tarea3_06.eps")
plt.show(1)
```

Tarea_3.py

Referencias

[1] Surender Baswana and Shahbaz Khan. Incremental algorithm for maintaining a dfs tree for undirected graphs. *Algorithmica*, 79(2):466–483, 2017.

[2] Patric RJ Östergård. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 120 (1-3):197–207, 2002.

[3] David J Pearce and Paul HJ Kelly. A dynamic topological sort algorithm for directed acyclic graphs. *Journal of Experimental Algorithmics (JEA)*, 11:1–7, 2007.

[4] Douglas R White and Stephen P Borgatti. Betweenness centrality measures for directed graphs. *Social networks*, 16(4):335–346, 1994.

[5] Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM (JACM)*, 49(3):289–317, 2002.