Práctica 3

Jose David Martinez Ruiz

8 de abril de 2018

1. Introducción

En esta tercera tarea se busca mejorar la clase creada en la tarea anterior. Para eso se incluyeron a la clase algoritmos empleados para grafos, de los cuales se busca de medir los tiempos de ejecución de esos algoritmos para así poder concluir sobre su complejidad computacional.

2. Desarrollo

Lo primero que se hizo fue adaptar las mejoras a la clase, las cuales constaban del algoritmo de Ford-Fulkerson y el algoritmo de Floyd-Warshall. El primero de estos dos algoritmos lo que hace es regresar el flujo máximo que puede haber entre un par de nodos a través de todos los caminos posibles que hay entre ese par de nodos, en caso de que no haya ningún camino entre los nodos, el algoritmo regresará el valor de cero. El segundo algoritmo, el algoritmo de Floyd-Warshall, lo que hace es calcular el camino más corto para cada par de nodos que hay en el grafo, es decir crear un directorio donde las entradas son cada par de nodos y su definición es el valor del camino más corto entre ellos.

Para poder adaptar estos algoritmos al código, lo que se hizo fue cambiar un poco la estructura del código que se nos fue proporcionado y también adecuar la forma en la que las variables de la clase son recibidas en los algoritmos. Como nos interesa saber la identidad de un nodo dentro de la lista de nodos para que a la hora de ejecutar los algoritmos puedan funcionar correctamente, se creó una función con la cual recibiendo la posición en donde se encuentra el nodo la función regresara la posición de ese nodo en la lista donde se encuentra. De una forma similar se hizo para poder obtener la posición de una arista dentro de la lista de aristas.

```
def index_v(p, N):
    for i in range(len(N)):
        if p[0] is N[i][0] and p[1] is N[i][1]:
            return(i)

def index_A(p, q, A, N):
    for i in range(len(A)):
        if N[p][0] is A[i][0] and N[p][1] is A[i][1]
        and N[q][0] is A[i][2] and N[q][1] is A[i][3]:
            return(i)
    return(1.1)
```

La última función lo que hace es regresar la posición de una arista en particular en la lista de las aristas, en caso de que se le pida la identidad de dos nodos que no están conectados, lo que hará es regresar un valor que no es entero para que al momento de querer realizar operaciones no pueda realizar ninguna operación y pase a verificar otro par de nodos.

Como estos algoritmos funcionan con grafos ponderados y por cómo estaban estructurados recibían como parámetro la lista de los pesos, se modificó para que no lo necesitara ya que la lista de pesos es una variable que se encuentra dentro de la clase.

Con estas adecuaciones ya se puede empezar a probar los algoritmos y verificar que funcionen

bien. Para eso se creó un grafo pequeño ponderado y también un grado similar pero dirigido para así verificar el funcionamiento de los algoritmos y corregir en caso de fuera necesario. Al estar todo en orden y funcionando se procedió a ejecutar el programa con diferente número de nodos varias veces para así obtener una muestra de cada uno. Dichas muestras son del tiempo de ejecución del algoritmo con esa cantidad de nodos, se hizo eso para así poder estimar la complejidad computacional de esos algoritmos. Se tomaron veinte muestras con cinco nodos, luego con diez y así hasta cincuenta.

Para realizar Teniendo ya todos los datos necesarios se procedió a verificar si los datos eran normales o no, para así saber cómo tratar a los datos. Se realizó la prueba de normalidad de Shapiro para los datos recolectados y la prueba nos arrojó que tanto los datos tomados de grados ponderados como los grafos ponderados y dirigidos no tienen una distribución normal.

Debido a que los datos no eran normales, se hizo un boxplot para así poder observar de mejor manera el comportamiento de los tiempos de las muestras dependiendo del número de nodos, esto se hizo tanto para grafos no dirigidos como para grafos dirigidos para ambos algoritmos.

Para los grafos no dirigidos estos son los boxplot obtenidos.

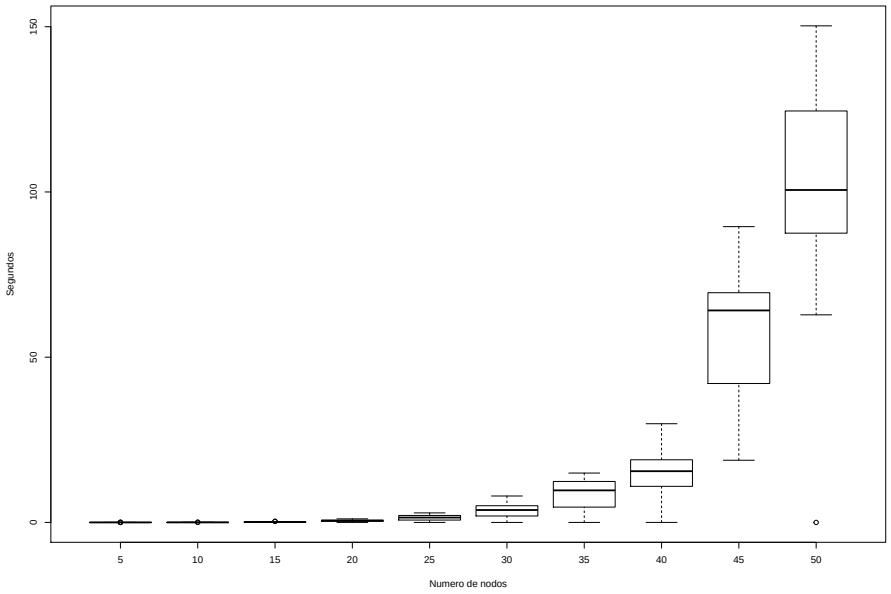

Figura 1: Algoritmo de Ford-Fulkerson.

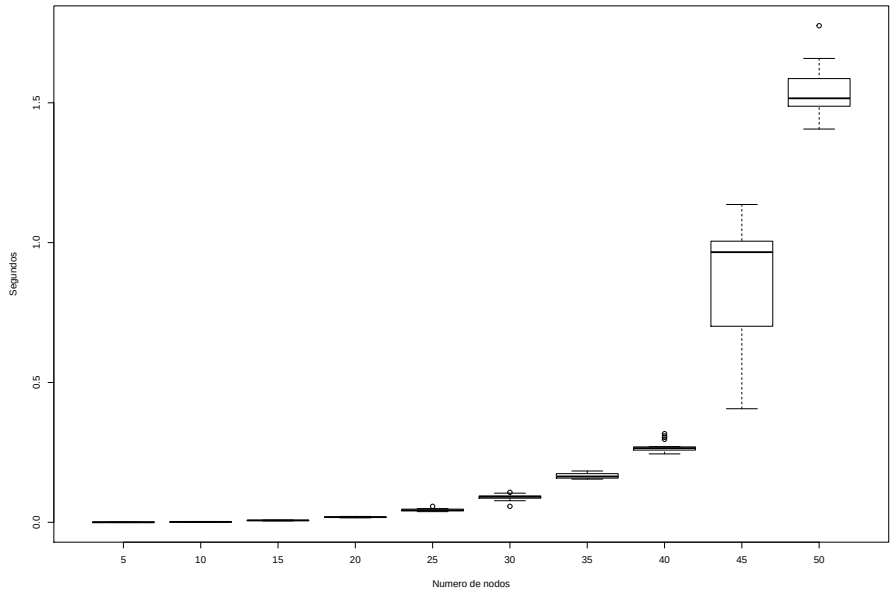

Figura 2: Algoritmo de Floyd-Warshall.

Para los grafos dirigidos estos son los boxplot obtenidos.

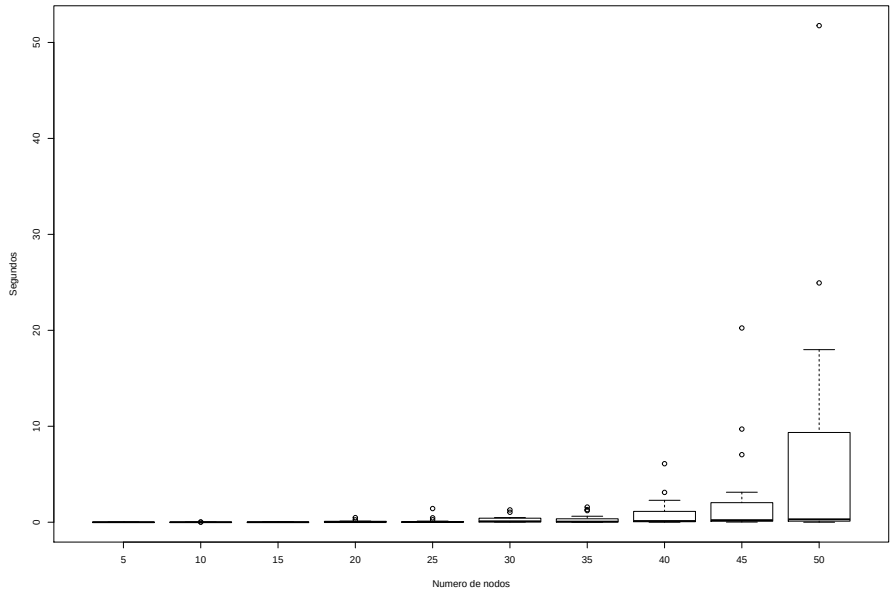

Figura 3: Algoritmo de Ford-Fulkerson.

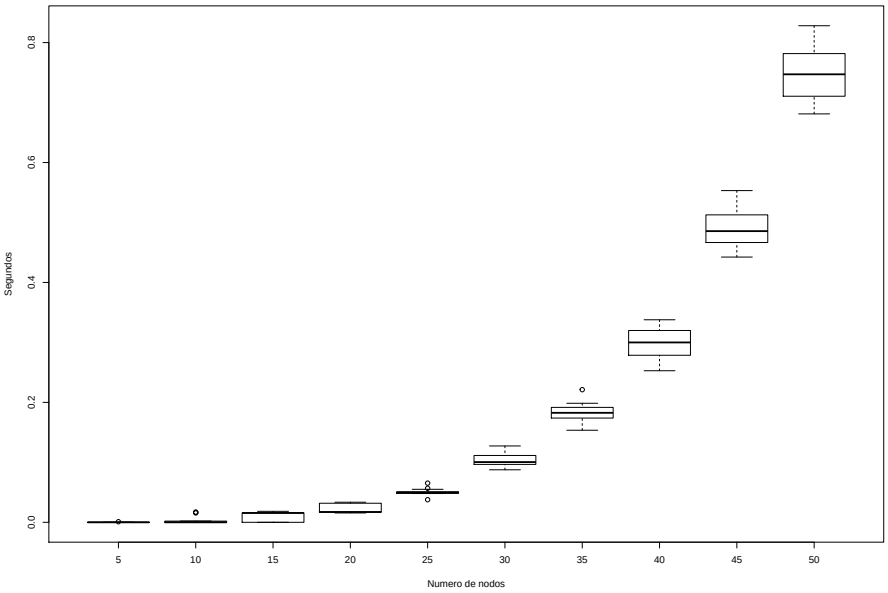

Figura 4: Algoritmo de Floyd-Warshall.

3. Conclusiones

Al mejorar el código agregando los algoritmos se amplían los problemas que se pueden resolver con grafos, además de que se puede conocer más sobre los grafos que se crean.
La complejidad computacional de estos algoritmos es polinomial, de n^2 para para el algoritmo de Ford-Fulkerson y n^3 para el algoritmo de Floyd-Warshall, sin embargo la forma de cómo se comportan los tiempos para estos algortimos no parecen comportarse del todo como la complejidad computacional que tienen realmente, esto puede deberse a la forma en la que se aplican los algoritmos en el código, como las funcionen auxiliares que se usan en los algoritmos, éstas puede que requieran de un poco más de tiempo del necesario, o también que haya faltado más muestras para que se adecuara mejor a la forma polinomial que su complejidad tiene .

Referencias

[1] Wikipedia. La enciclopedia libre. Algoritmo de Floyd-Warshall. Diciembre 2017. https://es.wikipedia.org/wiki/Algoritmo_de_Floyd-Warshall

[2] Wikipedia. The free encyclopedia. Ford–Fulkerson algorithm. Marzo 2018. https://en.wikipedia.org/wiki/Ford-Fulkerson_algorithm

[3] David Martinez. Tarea 2 Optimzación de flujo en redes. Marzo 2018. https://github.com/DavidM0413/Flujo-en-Redes/blob/master/practica2/p2