Complejidad asintótica experimental

Brenda Yaneth Sotelo Benítez
5705

3 de junio de 2019

En este trabajo se realiza la medición de tiempo de ejecución de cinco algoritmos con cinco grafos distintos, con la implementación de la librería Networkx [1] y Matplotlib [2] de `Python` [3] para la generación de grafos y guardar imágenes en el formato *eps* respectivamente. El código empleado se obtuvo consultando documentación oficial [4] y los grafos utilizados se encuentran en el repositorio de Sotelo [5] que fueron modificados para cumplir con los requerimientos que cada algoritmo tiene sobre sus datos de entrada.

Cada ejecución se repite una cantidad suficiente de veces para que el tiempo total de ejecución del conjunto de réplicas sea mayor a 5 segundos, posteriormente se repite la medición del conjunto de réplicas 30 veces en total.

Se presenta una breve descripción de cada algoritmo seleccionado, visualización de los grafos, el histograma de comparación de tiempos promedio para cada grafo, fragmentos relevantes de código y dos gráficas de dispersión.

Las pruebas se han realizado en una PC con procesador Intel Core i3 2.00 GHz y 8.00 GB de RAM.

Algoritmo 1

El algoritmo `maximum_flow` recibe como parámetros principales un grafo simple G dirigido o no dirigido, capacidad de los arcos, que en dado caso de no ser especificada se considera que tienen capacidad infinita, un nodo fuente s y un nodo sumidero t para el flujo.

Consiste en encontrar la cantidad máxima de flujo que puede circular desde s hasta t, por lo que devuelve como resultado el valor total de flujo máximo y el valor del flujo que pasó por cada arco.

La tabla 1 muestra la media y desviación estándar del tiempo de ejecución del conjunto de réplicas para cada grafo y en la figura 1 el histograma correspondiente al tipo de grafo y su tiempo promedio.

Grafo	Media	Desviación
1	5.5907	0.4611
2	7.5716	0.5386
3	12.8474	0.6974
4	9.6711	0.6482
5	5.0180	0.4612

Tabla 1: *Reporte de la media y desviación*

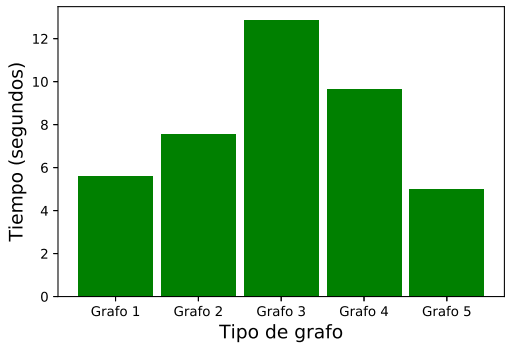
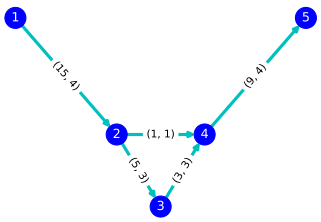


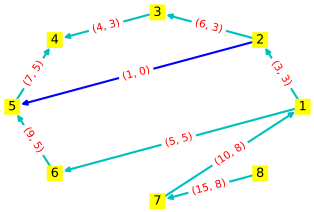
Figura 1: *Tiempo promedio para maximum_flow*

Se presenta a continuación en la figura 2 la solución para cada uno de los grafos donde en cada uno de ellos se muestra la ruta, la capacidad y el flujo que pasa a través de los arcos.

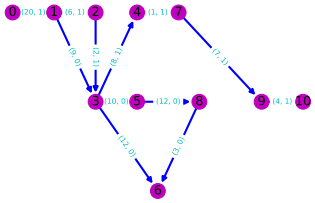
```
1 valor_flujo , flujo_max =nx.maximum_flow(G,1,5,capacity = 'peso')
```



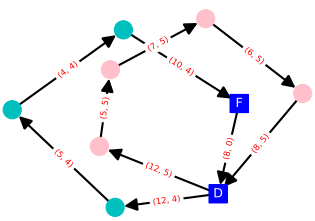
(a) Grafo 1



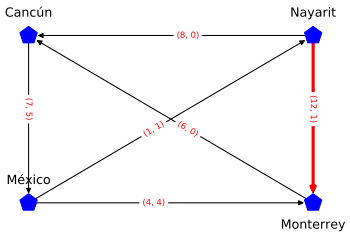
(b) Grafo 2



(c) Grafo 3



(d) Grafo 4



(e) Grafo 5

Figura 2: *Solución de maximum_flow*

Algoritmo 2

El algoritmo `all_shortest_paths` recibe como parámetros principales un grafo simple G dirigido o no dirigido, peso en los arcos, que en dado caso de no ser especificada se toma como peso, distancia o costo igual a 1 , un nodo inicial *source* y nodo final *target* para la ruta, asi como el método, es decir, el algoritmo que se utiliza para calcular las longitudes de la ruta: *dijkstra* o *bellman-ford*.

Consiste en buscar el camino más corto entre dos nodos, por lo que devuelve como resultado todas las rutas más cortas en el grafo.

La tabla 2 muestra la media y desviación estándar del tiempo de ejecución del conjunto de réplicas para cada grafo y en la figura 3 el histograma correspondiente al tipo de grafo y su tiempo promedio.

Grafo	Media	Desviación
1	6.0375	0.4586
2	10.5003	0.7112
3	14.9692	1.4172
4	10.6432	0.7121
5	9.3038	1.0171

Tabla 2: *Reporte de la media y desviación*

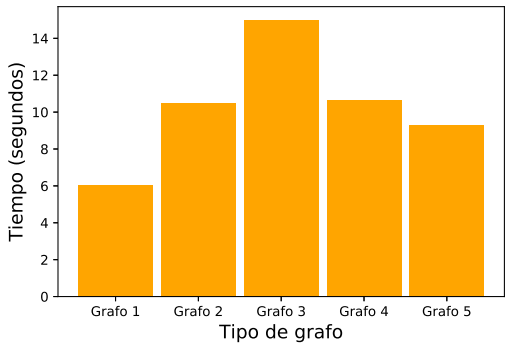
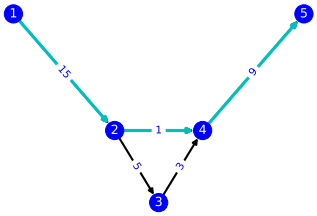


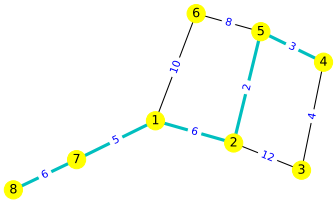
Figura 3: *Tiempo promedio para all_shortest_paths*

Se presenta a continuación en la figura 4 la solución para cada uno de los grafos, en cada uno de ellos se muestra la ruta más corta y el peso de cada uno de los arcos.

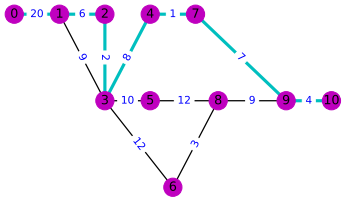
```
1 R = [p for p in nx.all_shortest_paths(G, source=1, target=5,weight=
      'peso')]
```



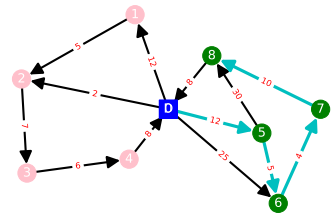
(a) Grafo 1



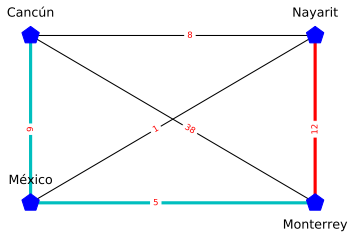
(b) Grafo 2



(c) Grafo 3



(d) Grafo 4



(e) Grafo 5

Figura 4: *Solución de all_shortest_paths*

Algoritmo 3

El algoritmo `minimum_spanning_tree` recibe como parámetros principales un grafo simple G dirigido o no dirigido, peso en los arcos, asi como el método, es decir, el algoritmo que se utiliza para encontrar un árbol de expansión mínima: *kruskal*, *bprim* o *boruvka*. El algoritmo predeterminado es *kruskal*.

Devuelve como resultado un árbol o bosque de expansión mínima en un grafo.

La tabla 3 muestra la media y desviación estándar del tiempo de ejecución del conjunto de réplicas para cada grafo y en la figura 5 el histograma correspondiente al tipo de grafo y su tiempo promedio.

Grafo	Media	Desviación
1	6.3368	0.352
2	7.2527	0.4135
3	9.9281	0.6668
4	8.3248	0.4654
5	4.3935	0.3181

Tabla 3: *Reporte de la media y desviación*

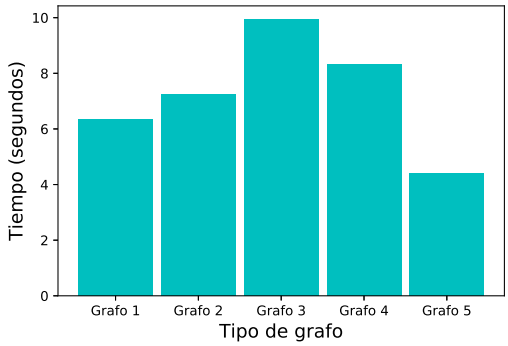
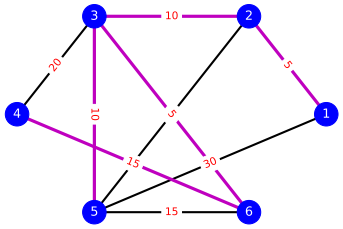


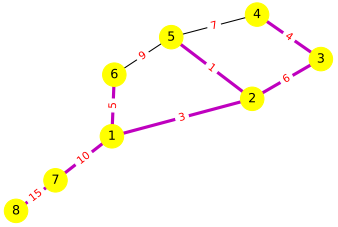
Figura 5: *Tiempo promedio para minimum_spanning_tree*

Se presenta a continuación en la figura 6 la solución para cada uno de los grafos, en cada uno de ellos se muestra el camino que sigue el árbol de expansión.

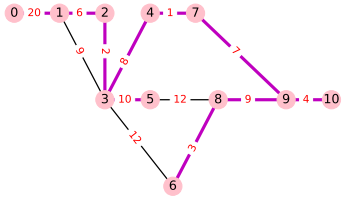
```
1 Tm = nx.minimum_spanning_tree(G, algorithm='kruskal')
```



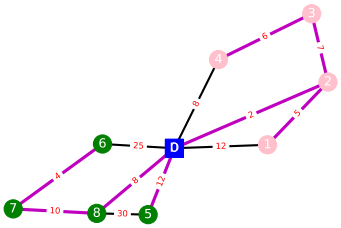
(a) Grafo 1



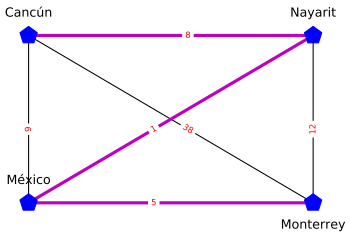
(b) Grafo 2



(c) Grafo 3



(d) Grafo 4



(e) Grafo 5

Figura 6: *Solución de minimum_spanning_tree*

Algoritmo 4

El algoritmo `greedy_color` recibe como parámetros principales un grafo simple G dirigido o no dirigido, una estrategia para el orden de coloración de los nodos y un algoritmo de intercambio.

Consiste en colorear un grafo con la menor cantidad de colores posible, donde para cada vecino de un nodo no puede tener el mismo color que el mismo.

La tabla 4 muestra la media y desviación estándar del tiempo de ejecución del conjunto de réplicas para cada grafo y en la figura 7 el histograma correspondiente al tipo de grafo y su tiempo promedio.

Grafo	Media	Desviación
1	3.4836	0.6221
2	5.3441	2.0836
3	6.2121	1.1141
4	5.3312	1.4626
5	4.0002	1.6867

Tabla 4: *Reporte de la media y desviación*

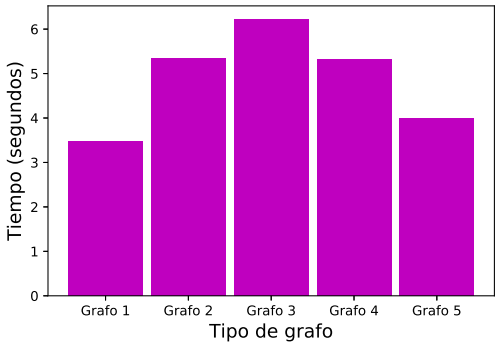
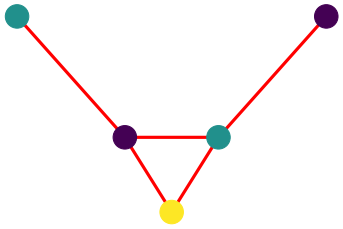


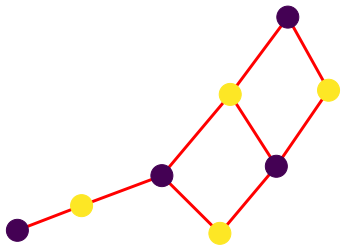
Figura 7: *Tiempo promedio para greedy_color*

Se presenta a continuación en la figura 8 la solución para cada uno de los grafos.

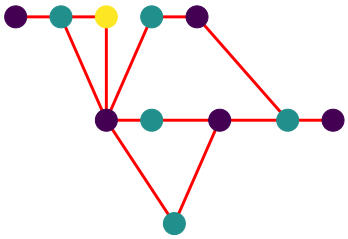
```
1        d = nx.coloring.greedy_color(G, strategy='largest_first')
```



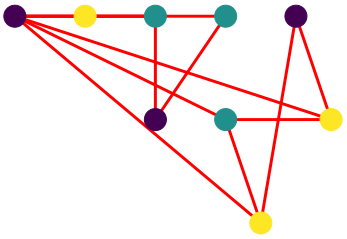
(a) Grafo 1



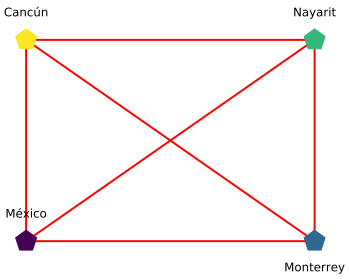
(b) Grafo 2



(c) Grafo 3



(d) Grafo 4



(e) Grafo 5

Figura 8: *Solución de greedy_color*

Algoritmo 5

El algoritmo `dfs_tree` recibe como parámetros principales un grafo simple G dirigido, un nodo fuente opcional para el inicio de la búsqueda en profundidad y la especificación del límite de búsqueda a profundidad.

Consiste en recorrer todos los nodos de un grafo comenzando de un nodo fuente y visitando a sus nodos adyacentes de tal manera que va formando un árbol a profundidad. Devuelve como resultado un árbol dirigido.

La tabla 5 muestra la media y desviación estándar del tiempo de ejecución del conjunto de réplicas para cada grafo y en la figura 9 el histograma correspondiente al tipo de grafo y su tiempo promedio.

Grafo	Media	Desviación
1	6.7802	0.4576
2	12.0012	1.4879
3	15.2492	0.6522
4	12.4167	1.4608
5	6.2011	0.5338

Tabla 5: *Reporte de la media y desviación*

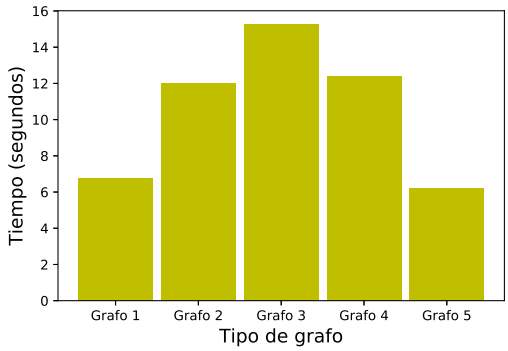
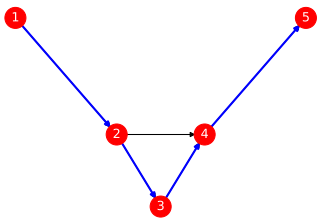


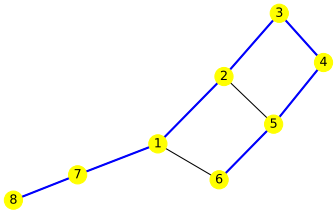
Figura 9: *Tiempo promedio para dfs_tree*

Se presenta a continuación en la figura 10 la solución para cada uno de los grafos, en cada uno de ellos se muestra el camino que sigue el algoritmo.

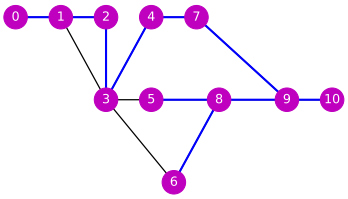
```
1        Tm = nx.dfs_tree(G, source=1)
```



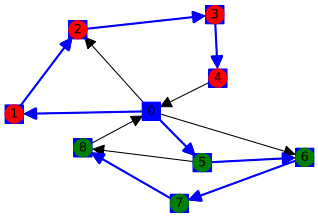
(a) Grafo 1



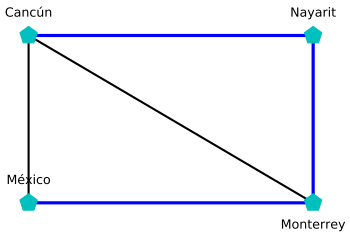
(b) Grafo 2



(c) Grafo 3



(d) Grafo 4



(e) Grafo 5

Figura 10: *Solución de dfs_tree*

En la figura 11 y figura 12 se muestra la gráfica de dispersión de cantidad de nodos y arcos contra el tiempo promedio de ejecución de cada uno de los grafos con los cinco tipos de algoritmos respectivamente. Cada forma de la gráfica representa un tipo de algoritmo.

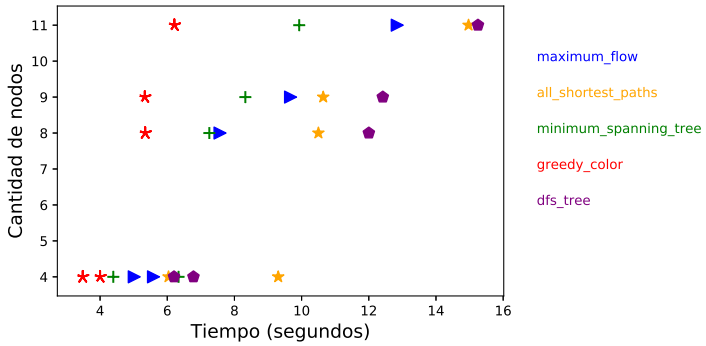


Figura 11: *Cantidad de nodos contra tiempos*

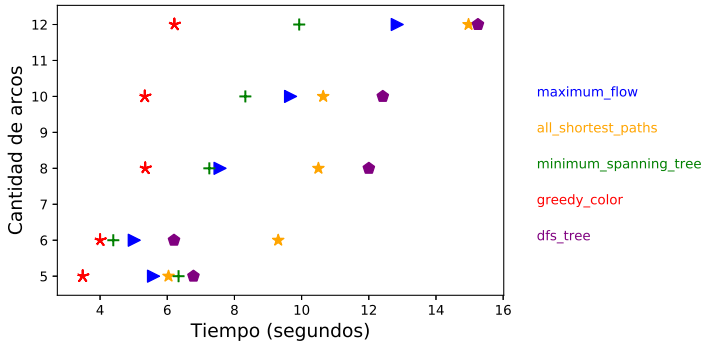


Figura 12: *Cantidad de arcos contra tiempos*

Referencias

[1] NetworkX developers Versión 2.0. `https://networkx.github.io/`.

[2] The Matplotlib development team Versión 3.0.2. `https://matplotlib.org/`.

[3] Python Software Foundation Versión 3.7.2. `https://www.python.org/`.

[4] NetworkX developers con última actualización el 19 de Septiembre 2018. `https://networkx.github.io/documentation/networkx-1.10/reference/drawing.html`.

[5] Sotelo B. Repositorio optimización flujo en redes. `https://github.com/BrendaSotelo/Flujo_Redes_BSotelo`.