SALVO a) NO ILUSTRAR UNION-FIND
y b) Tiempo de corrida
muy buen trabajo. Tiene extras en JUPYTER NB
GRACIAS

90 pts
+ 10 extra
100 pts

EIF-203 Estructuras discretas para informática

Investigación algoritmo de Kruskal

Isaac Fabián Palma Medina 1-1865-0422

NRC-41713

Declaración jurada

Declaro de manera jurada que este trabajo fue elaborado por mi persona de manera estrictamente individual y que las fuentes usadas son las declaradas en la lámina de referencias, las cuales sirvieron de base pero no fueron usadas como copia literal.

Referencias

1. Bari, A. (2018, 9 febrero). *3.5 Prims and Kruskals Algorithms - Greedy Method*. YouTube. https://www.youtube.com/watch?v=4ZlRH0eK-qQ
2. Chaudhary, T. (2018, 28 septiembre). *Bubble sorting in dictionary in python*. Stack Overflow. https://stackoverflow.com/questions/52551033/bubble-sorting-in-dictionary-in-python
3. Colaboradores de Wikipedia. (2020, 16 noviembre). *Algoritmo de Kruskal*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/w/index.php?title=Algoritmo_de_Kruskal&oldid=130982430
4. Colaboradores de Wikipedia. (2022a, mayo 27). *Timsort*. Wikipedia. https://en.wikipedia.org/wiki/Timsort
5. Colaboradores de Wikipedia. (2022b, junio 5). *Disjoint-set data structure*. Wikipedia. https://en.wikipedia.org/wiki/Disjoint-set_data_structure
6. Fiset, W. (2017, 8 abril). *Union Find Kruskal's Algorithm*. YouTube. https://www.youtube.com/watch?v=JZBQLXgSGfs
7. HackerEarth. (2016, 25 abril). *Minimum Spanning Tree Tutorials & Notes | Algorithms*. https://www.hackerearth.com/practice/algorithms/graphs/minimum-spanning-tree/tutorial/
8. Martineau. (2013, 21 enero). *What is the complexity of the sorted() function?* Stack Overflow. https://stackoverflow.com/questions/14434490/what-is-the-complexity-of-the-sorted-function
9. Matalka, L. (2021, 22 diciembre). *Sorting a Dictionary in Python - Towards Data Science*. Medium. https://towardsdatascience.com/sorting-a-dictionary-in-python-4280451e1637
10. NetworkX. (2022). *Shortest_path — documentation*. https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.shortest_paths.generic.shortest_path.html
11. Scanftree.com. (2022). *Kruskal's algorithm*. https://scanftree.com/Data_Structure/kruskal's-algorithm
12. Sedgewick, R., & Wayne, K. (2021). *Algorithms*. Algorithms, 4th Edition. https://algs4.cs.princeton.edu/home/
13. Sryheni, S. (2020, 19 octubre). *Determine Whether Two Nodes in a Graph Are Connected*. Baeldung on Computer Science. https://www.baeldung.com/cs/check-if-two-nodes-are-connected
14. Universitat Politècnica de València. (2013, 19 febrero). *S4.4- Algoritmo de Kruskal | | UPV*. YouTube. https://www.youtube.com/watch?v=YHzllcQpEdA
15. Uría, B. (2022, 30 marzo). *Qué son y cómo utilizar Lambdas en Python*. Medium. https://borjauria.es/que-son-y-como-utilizar-lambdas-en-python-4d1d168e2f90

Noción

- El **algoritmo de Kruskal** es usado en la tarea de encontrar el *minimum spanning tree* (MST) de un grafo (no dirigido) conectado.[1] [3] [11] [12]
- Un MST se trata de un subconjunto (conectado) de todos los vértices de un grafo, relacionados por medio arcos cuya **suma de pesos debe ser la menor posible**. [3] [7]
- El algoritmo toma su nombre de **Joseph Kruskal** (matemático y estadístico estadounidense), y fue publicado en 1956. [3]
- Se trata de un algoritmo *greedy*. [11] Puede explicar por qué?

Pasos generales

1. Primeramente, los pesos de los arcos del grafo son ordenados de manera **ascendente**, visitando cada par de vértices y el arco que los conecta. [1] [6] [12] [14]
2. Seguidamente se visualizan (escriben) los **vértices sin sus arcos**. [1] [6]
3. Finalmente, son añadidos los arcos según el orden en el que fueron escritos, además de verificar que **no se creen ciclos** (si un arco generara un ciclo este se descarta y se continua con el siguiente), hasta conseguir un grafo conectado. [1] [6] [14] Es importante denotar que para la cantidad de vértices (|V|), son necesarios un máximo de |V| - 1 arcos.

Bajo el proceso anterior la suma del peso de los arcos será la mínima posible, llegando a un **MST**. [7]

Sobre el paso 3 y la generación de ciclos

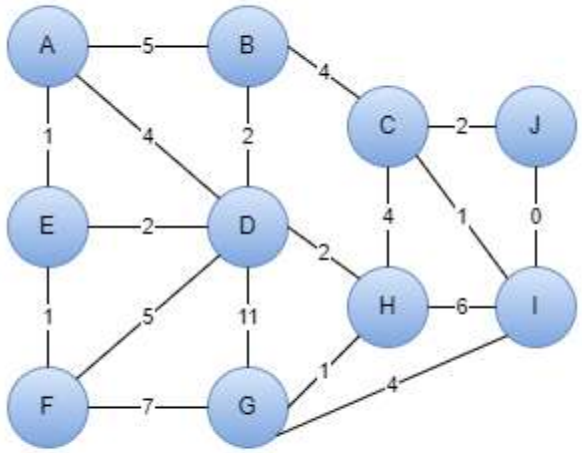
Al tratar de añadir nuevos arcos al grafo se podrían generar **ciclos indeseados**, dicho problema puede ser detectado gracias al algoritmo *Union-Find*. El algoritmo puede ser separado, por un lado una parte de búsqueda (*find*), encargada de determinar si los **elementos del arco a incluir se encuentran en el mismo subconjunto** (eso se desea evitar). [5][6] [12]

Además, de la parte encargada de la unión (*union*) de nuevos subconjuntos, dando lugar a **uno nuevo**. [5]

Ejemplo ilustrativo

Dado el siguiente grafo, se hace el ordenamiento de los pesos de sus arcos, siguiendo un orden ascendente, en caso de tener el mismo peso el orden es arbitrario. [1][6]

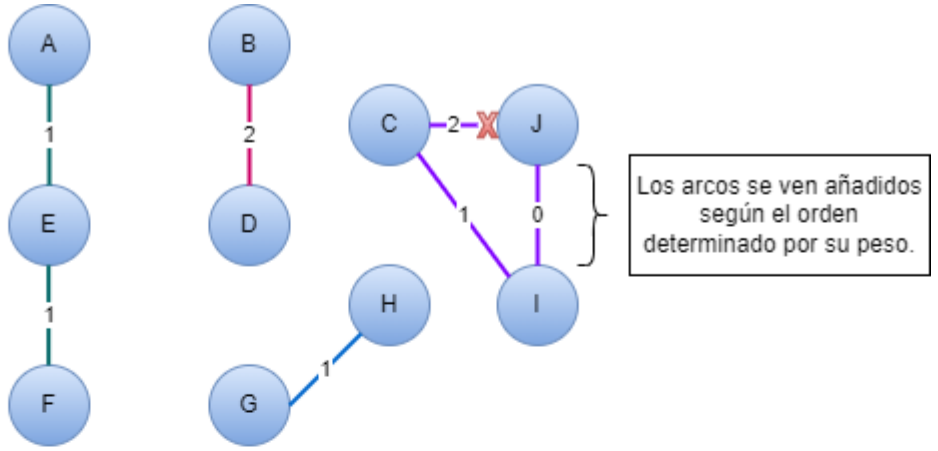
Fuente del grafo
WilliamFiset. (2017, April 07). *Union Find Kruskal's Algorithm*. Youtube. Recuperado de https://www.youtube.com/watch?v=JZBQLXgSGfs



Sorted edges

Edge	IJ	AE	CI	EF	GH	BD	CJ	DE	DH	AD	BC	CH	GI	AB	DF	HI	FG	DG
Cost	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11

Posteriormente se hace el ingreso de los arcos según la tabla, verificando que su inserción no resulte en problemas (ciclos). [1][6]



Sorted edges

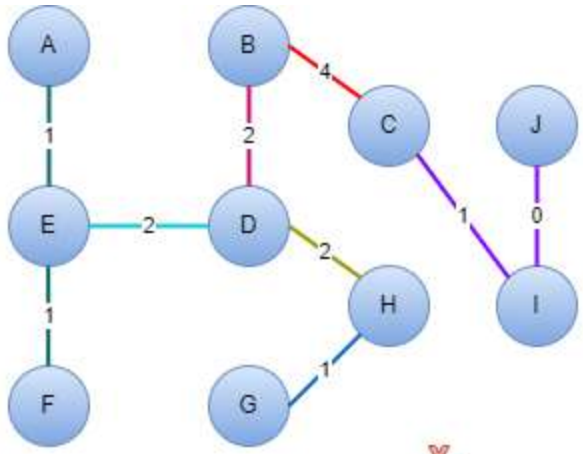
Edge	IJ	AE	CI	EF	GH	BD	CJ	DE	DH	AD	BC	CH	GI	AB	DF	HI	FG	DG
Cost	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11

I se relaciona con otro nodo. Se hace la pregunta: ¿C pertenece al mismo subgrafo? (Implementación del *find*) Al ser la respuesta "no", se añade ese arco (implementación de union).

C y J se encuentran en el mismo subgrafo (lo determina el *find*). La unión de estos implicaría la generación de un ciclo, por lo que se omite.

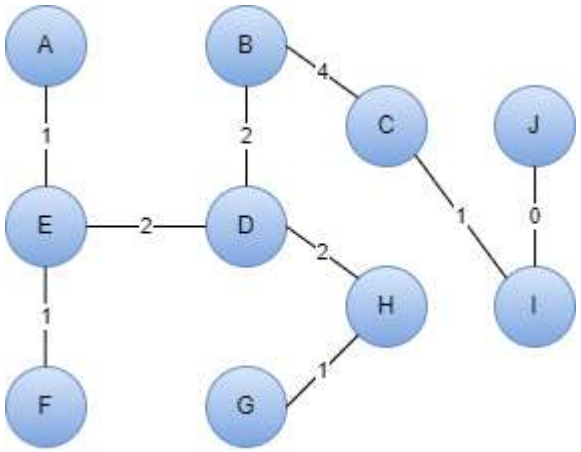
El proceso es realizado siempre, ya que un nodo pertenece a su propio subgrafo, el cual es distinto del subgrafo del nodo al que se quiere relacionar.

Se finaliza al llegar a un grafo conectado.

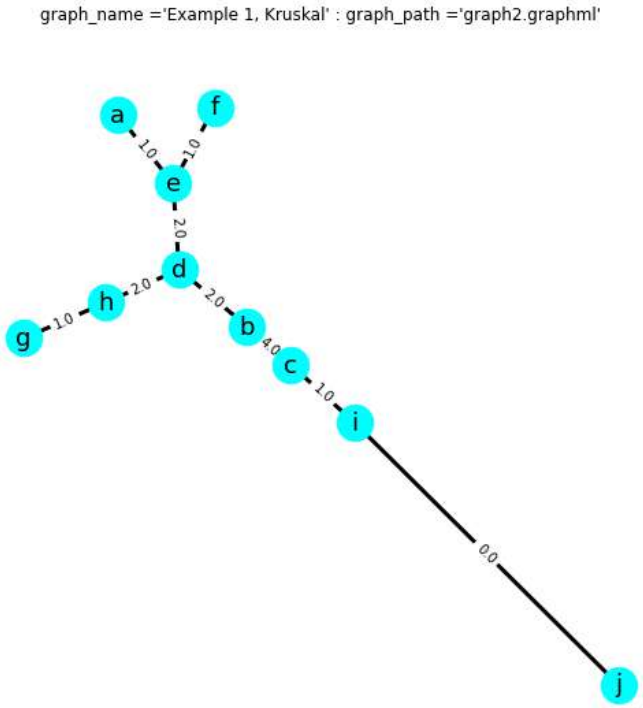


Sorted edges																		
Edge	IJ	AE	CI ✓	EF ✓	GH	BD	CJ ✗	DF ✓	DH ✓	AD ✗	BC ✓	CH	GI	AB	DF	HI	FG	DG
Cost	0	1	1	1	1	2	2	2	2	4	4	4	4	5	5	6	7	11

Respuesta según el proceso a mano

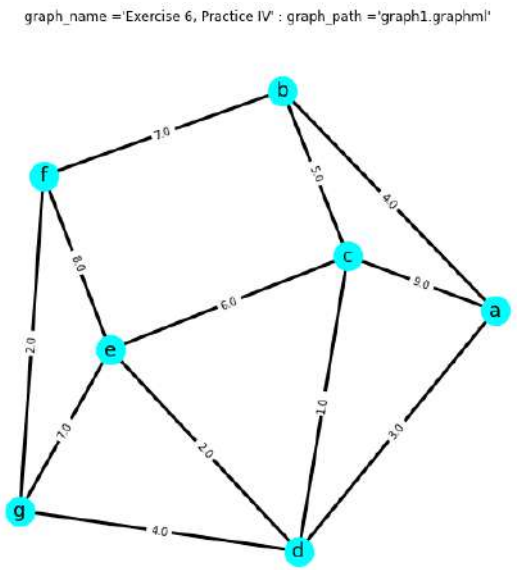


Respuesta según una implementación en Python (se anexa)



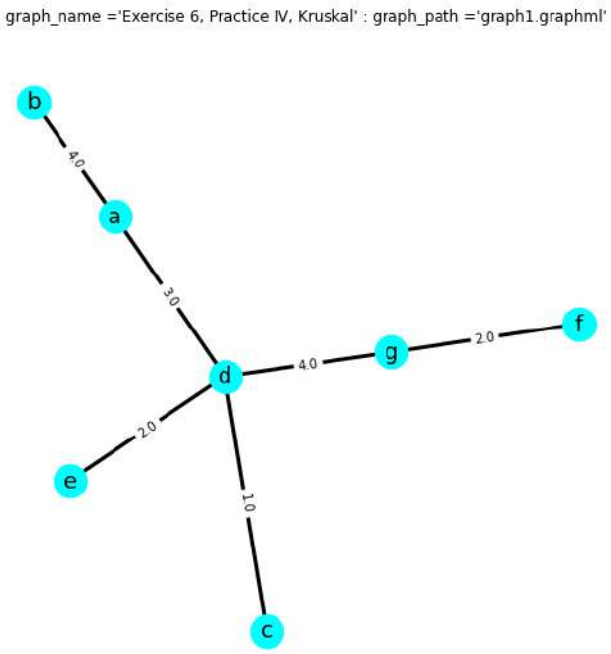
Ejemplo alternativo

Ejemplo 6 Práctica IV (mostrado en Python)



Loría, C. (2022). *Práctica_IV_GrafosDiscretas(Examen)_I_2022.pdf.* Recuperado de: https://drive.google.com/file/d/1Pr9RdVH3u8YoB4ZrpoT-wLkAwfm5BhNp/view

Respuesta según una implementación en Python (se anexa)



Tiempo de corrida

El análisis de tiempo puede ser logrado mediante la separación de la noción del algoritmo en partes, se identifica el ordenamiento del conjunto de arcos (por peso) así como la constante llamada a *Union-Find*, para verificar si un nodo ya pertenece a un cierto subgrafo. Las implementaciones de *Union-Find* son varias pero la lógica es la misma, y se apela a la hecho de que podría existir de una manera previa de llegar entre los nodos (pertenencia a otro subgrafo, según un padre). [4] [6] [12]

Tiempo de corrida

¿Qué es n?

IMPRECISO si n = |V|
El Kruskal es sobre arcos no vértices

En cuanto al tiempo de corrida del ordenamiento, tomando por ejemplo a la función *sorted* de Python este una complejidad de $O(nlog(n))$, esto debido a la implementación "en el fondo" de un algoritmo denominado *Timsort,* cuyo tiempo en el peor caso es el mencionado. Lo anterior puede ser una manera de ordenar un conjunto, en este caso el conjunto sería el de los pesos de los arcos. [4] [8]

En cuanto a la parte del *Union-Find* se visita la bibliografía sobre las *disjoint-set data structures,* para este caso se implementan determinadas operaciones las cuales permiten el control sobre conjuntos, en especial la posibilidad de fusionar conjuntos (*union* y creación de un nuevo *set*) y de averiguar si dos elementos pertenecen al mismo conjunto (*find*), bajo el resultado de lo anterior se da la lógica ya expuesta; estas verificaciones tienen una complejidad en el peor caso de $O(n)$. [5]

Tiempo de corrida

Bajo lo anterior y tomando en cuenta que los tiempos son $O(nlog(n))$ y $O(n)$, se puede decir:

arcos ¿Qué es n? vértices

$$T_{Kruskal} = O(nlog(n)) + O(n) \Rightarrow f_1 + f_2 \sim O(\max(f_1, f_2))$$

$$T_{Kruskal} = O\left(\max\left(O(nlog(n)), O(n)\right)\right), por\ teorema\ de\ suma$$

$$T_{Kruskal} = O(nlog(n)), siendo\ n\ el\ número\ de\ vértices$$

IMPRECISO

arcos