TECNOLÓGICO DE MONTERREY®

TC1031 – PROGRAMACIÓN DE ESTRUCTURAS DE DATOS Y ALGORITMOS FUNDAMENTALES
GRUPO 5

Actividad 4.3
Actividad Integral
Grafos

Brenda Elena Saucedo González – A00829855

24 de noviembre de 2021

Investigación

Un grafo es un conjunto de vértices (nodos) y un conjunto de aristas (arcos) que los unen.

Gráficamente, se suelen representar los vértices como puntos en el plano y las aristas como segmentos que los unen.

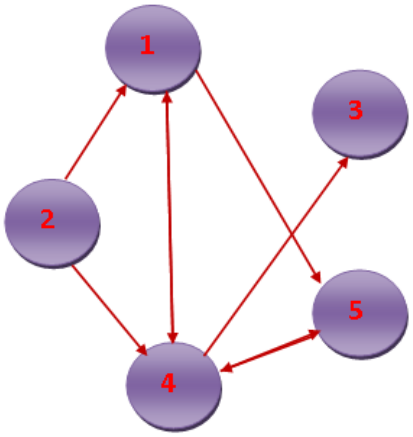

Ilustración 1. Grafo Dirigido.

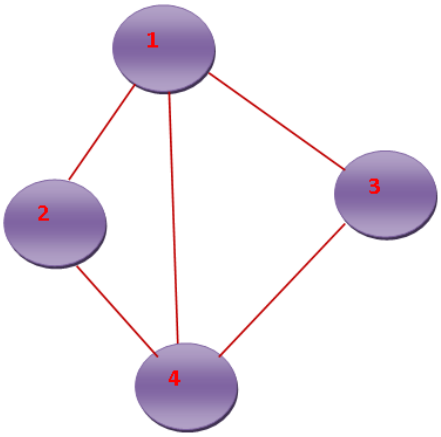

Ilustración 2. Grafo No Dirigido.

Los grafos son estructuras en las que podemos almacenar grandes cantidades de información, pero lo más importante es que permiten entender las conexiones y relaciones entre los datos que albergan, facilitando de esta forma la compresión al momento de resolver un problema.

Los grafos también tienen muchos tipos de aplicaciones, tanto de mapas como aplicaciones matemáticas, como el algoritmo de Floyd y Warshall, que tiene como objetivo buscar y resolver problemas sobre la búsqueda de caminos más cortos en grafos ponderados; el Teorema de los 4 colores, que consiste en colorear los vértices del grafo sin que haya el mismo color a la par del vértice; entre muchas otras aplicaciones en que la estructura del grafo permite su fácil y óptima implementación.

Análisis de Complejidad Computacional

Para la solución de la situación problema planteada, hicimos uso de la clase “GraphType” para poder implementar los grafos no dirigidos y no ponderados en la solución. En la clase “GraphType” tuvimos que hacer uso de diversos métodos para poder agregar nuevos elementos al grafo, sus respectivas conexiones con otros vértices, así como acceder a cierta información del grafo.

Métodos Implementados

El método **getIndex** tiene un orden de complejidad de O(v), ya que hace uso de un ciclo iterativo para buscar el índice de un vértice, en donde va a estar iterando “v” veces, ya que depende del número de vértices del grafo.

El método **addVertex** tiene un orden de complejidad de O(v), donde “v” es el número de vértices que tiene el grafo. Tiene este orden de complejidad debido a que primeramente se está llamando a un método (getIndex) que tiene un orden de complejidad de O(v), y luego se hace uso de un ciclo iterativo que depende de la misma cantidad de vértices “v”. Por esta razón, al ser procesos independientes, el método tiene un orden de complejidad de O(v) en el peor de los caso.

El método **addEdge** tiene un orden de complejidad de O(v) en el peor de los casos, ya que se llaman dos veces de forma independiente a un método (getIndex) que tiene un orden de complejidad de O(v).

El método **getAdjacents** tiene un orden de complejidad de O(v), ya que en un inicio llama a un método (getIndex) que tiene un orden de complejidad de O(v), y luego se hace uso de un ciclo iterativo que depende de la misma cantidad de vértices, es decir, va a estar iterando “v” veces. Al ser procesos independientes, se obtiene que el método getAdjacents tiene un orden de complejidad de O(v) en el peor de los casos.

El método **MNPsearch** tiene un orden de complejidad de O(v+e(e+n)), ya que primeramente se llama a un método (getAdjacents) que tiene un orden de complejidad de O(v), luego se hace uso de un ciclo iterativo que depende de la cantidad de arcos “e” de un vértice en específico, es decir, va a estar iterando “e” veces. Finalmente, dentro de este último ciclo, se hace uso de la recursividad, la cual depende de la cantidad de arcos “e” de un vértice y de mnp “n”, es decir, tiene un orden de complejidad de O(e+n), y en conjunto con el ciclo, O(e(e+n)). Por esta razón, al juntar todas estas variables, este método llega a tener un orden de complejidad de O(v+e(e+n)) en el peor de los casos.

El método **MNP** tiene un orden de complejidad de O(v+e(e+n)) en el peor de los casos, ya que primeramente se llama a un método (getIndex) que tiene un orden de complejidad de O(v), y luego se hace uso de un ciclo iterativo que depende de la misma cantidad de vértices “v”. Finalmente, se llama a un método (MNPsearch) que tiene un orden complejidad de O(v+e(e+n)). Por esta razón, al ser procesos independientes, el proceso de mayor complejidad es el de O(v+e(e+n)).

Reflexión

En cuanto a la investigación realizada, se puede observar claramente como ciertos aspectos del uso de los grafos, permiten encontrar una solución mucho más eficiente a la situación problema planteada que otro tipo de estructuras de datos.

Por el tipo de estructura de los grafos, estos nos facilitan el almacenamiento de datos, inclusive guardar la conexión que tienen con otros datos almacenados en el mismo, lo que nos facilita la comprensión de las relaciones que existe entre sus elementos.

Por esto mismo, en situaciones problemas de esta misma naturaleza, en donde se tienen que almacenar elementos con sus respectivas conexiones, la solución se vuelve más eficiente y óptima al hacer uso de los grafos, ya que su estructura posibilita la reducción de la complejidad del algoritmo, sobre todo para permitir agregar, almacenar y acceder más fácilmente a sus conexiones con otros elementos (arco).

Referencias

mbalsells. (2019). *Vocabulario Teoría de Grafos*. Noviembre 17, 2021, de Aprende Programación Competitiva. Sitio web: https://aprende.olimpiada-informatica.org/algoritmia-grafos

Redacción BDM. (2020). *Los Grafos y el Big Data*. Noviembre 17, 2021, de Big Data Magazine Sitio web: https://bigdatamagazine.es/los-grafos-y-el-big-data

uniioalfaro. (2011). importancia de los grafos. Noviembre 17, 2021, de Luis Alfredo Alfaro. Sitio web: https://uniioalfaro.wordpress.com/2011/03/03/importancia-de-los-grafos/

Anónimo. (2020). La utilidad y aplicación de los Grafos y Sistemas de Información Geográfica. Noviembre 17, 2021, de Delfino. Sitio web: https://delfino.cr/2020/10/la-utilidad-y-aplicacion-de-los-grafos-y-sistemas-de-informacion-geografica