UNIVERSIDAD MARIANO GALVEZ DE GUATEMALA

INGENIERIA EN SISTEMAS DE INFORMACIÓN

PROGRAMACIÓN III

Catedrático: Ing., José Luis Xiloj

Integrantes:	**Carné**	**Participación**
Angela Lolita Villalta García	1590-22-15807	100%
Ashley Deydania Arredondo Salazar	1590-21-7193	100%
Juan Pablo López Montoya	1590-16-11594	100%
Ludim Abisai Agreda Guzmán	1590-11-6130	100%
Jonathan Alexander Batres Rivas	1590-22-10252	100%
Sergio José Granados Mejía	1590-19-19917	100%

Sección: B

INDICE

INTRODUCCIÓN

Los grafos son estructuras de datos fundamentales en la programación que permiten representar y analizar relaciones entre objetos o entidades. Un grafo está compuesto por un conjunto de vértices (o nodos) y un conjunto de aristas (o conexiones) entre estos vértices. Los grafos son utilizados en una variedad de aplicaciones, desde la representación de redes sociales hasta la modelización de sistemas complejos en biología y logística. Existen varios tipos de grafos, cada uno con características y aplicaciones específicas. En este trabajo, se explorarán estos tipos y se discutirán sus usos y funciones en la programación y la gestión de sistemas. La versatilidad y capacidad de los grafos para modelar y resolver una amplia gama de problemas los convierten en una herramienta invaluable para representar relaciones complejas.

Definición

Es una estructura matemática que consiste en un conjunto de elementos llamados **vértices** (también conocidos como **nodos**) y un conjunto de conexiones entre pares de vértices, llamadas **aristas.**

En programación un grafo es un conjunto de puntos (llamados **nodos**) que están conectados por líneas (llamadas **aristas**). Cada nodo representa algo, como una tarea o un lugar, y las aristas muestran cómo están relacionados esos nodos.

Son como un mapa de conexiones entre cosas, y nos ayudan a resolver problemas y tomar decisiones, es como tener un mapa secreto para encontrar la mejor manera de hacer las cosas.

Son una herramienta sumamente esencial en programación debido a que su versatilidad y capacidad son ideales para moldear y resolver una amplia gama de problemas. Por lo tanto, los nodos pueden estar conectados entre sí de diversas formas y así convirtiéndose en una herramienta invaluable para representar relaciones complejas. En sí es una estructura de datos los cuales se utilizan para representar relaciones entre pares de objetos.

Tipos

Grafos Simples:

Es aquel que solo acepta una arista uniendo dos vértices diferentes. Esto es equivalente a decir que una solo arista diferente es lo que une dos vértices

Son la forma más básica de representar relaciones entre objetos o entidades en programación. Además de que no permite la existencia de múltiple conexiones entre los mismos nodos. Estas conexiones pueden ser **bidireccionales** lo que significa que se puede transitar en ambos sentidos, o **unidireccionales** permitiendo el flujo de información de una sola dirección.

La principal característica de los grafos simples es su sencillez y facilidad de implementación, lo que lo convierte en una herramienta muy útil para modelar una amplia variedad de problemas y relaciones en diferentes ámbitos de la informática

Multígrafos:

Este acepta más de una arista entre dos vértices y estas aristas se llaman múltiples o lazos. Es decir que, a diferencia de los grafos simples, donde sólo puede haber una única arista entre dos vértices, en un multígrafo pueden haber varias conexiones distintas entre los mismos puntos.

Los multígrafos son una variante de los grafos simples en los que puede haber más de una arista conectando dos vértices. Esto permite representar relaciones más complejas entre los elementos de un sistema, como por ejemplo diferentes tipos de interacción entre personas en una red social.

Grafos Completos:

Si cada par de vértices se une por una arista. Es un tipo de grafo en programación en el que cada par de nodos está conectado por una arista. Esto significa que todos los nodos están directamente conectados entre sí, sin pasar por otros nodos intermedios.

Estos grafos se caracterizan por tener un alto grado de conectividad y densidad, lo que los hace útiles en aplicaciones como redes sociales, análisis de tráfico, y optimización de rutas.

Los grafos completos tienen propiedades interesantes, como que el número de aristas es igual al número de combinaciones de 2 nodos (n(n-1) /2, donde n es el número de nodos). Además, todos los nodos tienen el mismo grado, es decir, el mismo número de conexiones. Esto los convierte en estructuras simétricas y balanceadas, lo que facilita su análisis y procesamiento.

Grafos Etiquetados:

Son aquellos que se han añadido un peso a las aristas o un etiquetado a los vértices. Son una variante de los grafos simples en la que cada arista o vértice tiene asociada una **etiqueta** que representa una cierta información adicional. Estas etiquetas pueden ser números, texto, o cualquier otro tipo de dato que se desee asociar a los elementos del grafo.

El hecho de tener etiquetas en los grafos permite representar información más compleja y realizar análisis más detallados.

Grafos Ponderados:

En un grafo ponderado, cada arista tiene asociado un valor numérico llamado **peso.** Estos pesos pueden representar diversas magnitudes, como costos, distancias, tiempos, capacidades, etc. La introducción de pesos en las aristas permite modelar de manera más precisa situaciones en las que las conexiones entre los nodos tienen alguna medida cuantitativa asociada.

Estos pesos influyen en la manera en que se recorren y se analizan los grafos, permitiendo encontrar caminos óptimos o realizar cálculos más precisos.

Para trabajar con grafos ponderados, se utilizan algoritmos específicos, como el algoritmo de Dijkstra o el algoritmo de Kruskal, que permiten encontrar caminos óptimos o árboles de expansión mínima.

Conceptos Básicos

Los grafos son una composición interesante de conjuntos de objetos que denominamos nodos. En ellos se almacena diferentes tipos de elementos o datos que podemos utilizar para procesar o conocer con fines específicos.

Adicionalmente estos nodos, suelen estar unidos o conectados a otros nodos a través de elementos que denominamos aristas.

Los nodos pertenecientes a un grafo pueden contener datos estructurada o no estructurada y al interrelacionarse con otros nodos producen relaciones interesantes que podemos analizar con diferentes finalidades.

Estos elementos son reconocidos por su capacidad de manejar altos volúmenes de datos y ser fácilmente procesados por motores de búsqueda o gestores de bases de datos orientados a grafos.

Aplicaciones o Usos

La tecnología de grafos ha revolucionado el mundo de los datos. Se han convertido en una de las principales herramientas para empresas y organismos públicos en la gestión de datos. Las capacidades sorprendentemente amplias de las bases de datos de grafo frente a los modelos relacionales tradicionales, permiten dar un salto de calidad gigante en materia analítica.

A continuación, conoceremos un poco más sobre las grandes posibilidades que la tecnología de grafos nos permite aprovechar y descubrirás 10 casos de uso REALES que han sido posibles gracias a los grafos.

Prevención del lavado de dinero

Las entidades financieras pueden crear modelos que analicen los datos de las transferencias, a los usuarios y sus vinculaciones para detectar patrones sospechosos y anticiparse a operaciones ilegales.

Gestión de Clientes

Gracias a los grafos, las empresas pueden contar con un análisis de 360 grados sobre sus clientes. Pueden crear bases de datos que contengan más que sus datos maestros. Es posible contemplar datos de transacciones, registros, preferencias y así poder crear y ofrecer propuestas de valor que sean de interés para el usuario y acciones que eviten la pérdida de clientes

Tecnología de grafos para las recomendaciones de productos

Gracias a las bases de datos de grafos se pueden crear modelos de sistemas de recomendación que consideren distintos factores. Desde las compras anteriores del usuario, los artículos más recientes con los que ha interactuado, entre otros. Estos sistemas brindan una visión rápida y amplia de lo que el usuario desea y necesita, incrementando los índices de conversión.

Redes neuronales de grafos

Estos modelos de recomendación aprovechan el aprendizaje automatizado para proveer soluciones de valor.

Posibilidades de Comunicación

La tecnología de grafos nos permite capturar en tiempo real, conexiones entre entidades de datos. También ayudan a detectar anomalías y comportamientos erróneos en los conjuntos de datos que se conectan en una red de TI para evitar de forma eficiente amenazas que pueden perjudicar la integridad de los datos. Numerosas empresas en la actualidad soportan algunos procesos de seguridad en este tipo de bases de datos, para contar con detección en tiempo real y visualización de datos para fortalecer la seguridad de sus datos.

Redes de Transporte

También es posible representar mediante un grafo los flujos que circulan a través de una red de transporte. Una red de transporte deberá tener uno o varios vértices origen y uno o varios vértices destinos. El resto de vértices serán vértices de transbordo.

El grafo que se presenta a continuación puede representar una red de distribución de agua entre diferentes puntos el vértice A es el origen de la red, y el vértice E el destino.

Propiedades de Los Grafos

Vértices (Nodos): Los vértices son los puntos o nodos del grafo. Representan entidades como personas en una red social, ciudades en un mapa, etc.

Aristas (Edges): Las aristas son las conexiones entre los vértices. Representan relaciones o caminos entre las entidades, como amistad entre personas o carreteras entre ciudades.

Grado (Degree): El grado de un vértice es el número de aristas incidentes a él. En un grafo dirigido, se distingue entre grado de entrada (in-degree) y grado de salida (out-degree).

Camino (Path): Un camino es una secuencia de vértices conectados por aristas. Un camino puede ser simple (sin vértices repetidos) o puede formar un ciclo.

Conectividad: Un grafo es conexo si hay un camino entre cualquier par de vértices. En grafos dirigidos, se puede hablar de fuerte conectividad (un camino en ambas direcciones) y débil conectividad (un camino en al menos una dirección).

Peso (Weight): En grafos ponderados, las aristas tienen un valor asociado llamado peso, que puede representar costos, distancias, etc.

Bipartito: Un grafo es bipartito si sus vértices pueden dividirse en dos conjuntos disjuntos tal que no hay aristas entre vértices del mismo conjunto.

Cíclico y Acíclico: Un grafo es cíclico si contiene al menos un ciclo; es acíclico si no contiene ciclos. Los grafos dirigidos acíclicos (DAG) son especialmente importantes en muchos algoritmos.

Características De Los Grafos

Representación:

Matriz de Adyacencia: Una matriz donde la entrada (i, j) es verdadera (o contiene el peso de la arista) si existe una arista del vértice i al vértice j.

Lista de Adyacencia: Una lista donde cada vértice tiene su propia lista de vértices adyacentes. Es más eficiente en términos de espacio para grafos dispersos.

Tipos de Grafos:

Grafos Dirigidos: Las aristas tienen una dirección, es decir, van de un vértice a otro.

Grafos No Dirigidos: Las aristas no tienen dirección y pueden ser recorridas en ambos sentidos.

Grafos Ponderados: Las aristas tienen pesos que representan algún costo o valor asociado.

Grafos No Ponderados: Todas las aristas tienen el mismo peso.

Aplicaciones:

Redes Sociales: Modelan relaciones entre usuarios.

Sistemas de Recomendación: Representan conexiones entre usuarios y productos**.**

Rutas y Mapas: Utilizados en sistemas de navegación y planificación de rutas.

Algoritmos de Búsqueda: Como el algoritmo de Dijkstra para encontrar el camino más corto.

Ventajas de Los Grafos

Flexibilidad: Los grafos pueden modelar una amplia variedad de sistemas y problemas reales.

Representación Natural de Relaciones: Las conexiones entre entidades se representan de manera intuitiva.

Eficiencia en Algoritmos de Caminos Cortos: Existen algoritmos optimizados (como Dijkstra, Bellman-Ford) para encontrar caminos óptimos en grafos.

Algoritmos de Búsqueda Eficientes: Los grafos permiten implementar búsquedas eficientes como BFS (Breadth-First Search) y DFS (Depth-First Search).

Análisis de Redes: Permiten el análisis de propiedades complejas de redes, como la centralidad y la detección de comunidades.

Desventajas de Los Grafos

Complejidad Espacial: Los grafos densos pueden requerir grandes cantidades de memoria, especialmente con la representación de matriz de adyacencia.

Complejidad Computacional: Algunos problemas en grafos son NP-completos, lo que significa que no hay soluciones eficientes conocidas para grafos grandes.

Dificultad en la Visualización: A medida que los grafos crecen, se vuelven difíciles de visualizar y entender.

Costos de Actualización: La modificación de grafos grandes puede ser costosa en términos de tiempo y recursos.

Dependencia del Tipo de Grafo: La eficiencia de los algoritmos puede variar significativamente según la estructura del grafo (dirigido, no dirigido, ponderado, etc.).

Los grafos son una herramienta poderosa y versátil en la programación y la informática. Su capacidad para modelar relaciones complejas y resolver una variedad de problemas hace que sean esenciales en muchas aplicaciones. Sin embargo, su uso también puede implicar desafíos significativos en términos de complejidad y manejo de datos, lo que requiere un enfoque cuidadoso en su implementación y análisis.

Función de Los Grafos En Programación

Son una herramienta esencial en programación por su versatilidad y capacidad para modelar y resolver muchos problemas. Algunas áreas clave donde los grafos son fundamentales en programación:

Redes y Comunicación:

Los grafos se utilizan para modelar redes de computadoras, sistemas de comunicación y enrutamiento de datos. Los nodos representan dispositivos (como computadoras o routers), y las aristas representan conexiones entre ellos.

Algoritmos de Búsqueda y Recorridos:

Los grafos se utilizan en algoritmos de búsqueda como el algoritmo de búsqueda en profundidad (DFS) y el algoritmo de búsqueda en anchura (BFS). Estos algoritmos son fundamentales para encontrar caminos, resolver problemas de conectividad, y más.

Grafos Dirigidos Acíclicos (DAGs):

Los DAGs se utilizan en sistemas de planificación y ejecución de tareas, donde las dependencias temporales entre actividades se modelan de manera eficiente.

Algoritmos de Caminos Mínimos:

Los grafos ponderados se utilizan en algoritmos como Dijkstra y Bellman-Ford para encontrar los caminos más cortos en redes, mapas o cualquier estructura donde se pueda asignar un peso a las conexiones.

Sistemas de Recomendación:

Los grafos se utilizan para modelar relaciones entre usuarios y elementos en sistemas de recomendación, como redes sociales o plataformas de comercio electrónico, para predecir preferencias y ofrecer recomendaciones personalizadas.

Modelado de Relaciones:

Los grafos se utilizan para representar relaciones y dependencias en una amplia variedad de campos, como bases de datos, análisis de dependencias en sistemas y modelado de estructuras de datos complejas.

Juegos y Estrategia:

En la inteligencia artificial, los grafos se utilizan para modelar árboles de juego en juegos estratégicos como el ajedrez o para tomar decisiones en entornos complejos.

Optimización y Flujo de Redes:

Los grafos se aplican en problemas de flujo máximo, como en la planificación de rutas de transporte, distribución de recursos y optimización de flujos en redes.

Compiladores y Análisis de Código:

En compiladores, los grafos se utilizan para representar la estructura sintáctica y semántica del código fuente, facilitando análisis y optimizaciones.

Representación de datos:

Los grafos son una forma de representar relaciones entre objetos. Esto puede ser útil en una amplia gama de aplicaciones, como redes sociales, sistemas de transporte, análisis de redes, y más.

Los grafos son una herramienta poderosa y versátil en programación que se aplica en una amplia variedad de problemas y dominios, lo que los convierte en una parte esencial del repertorio de cualquier programador.

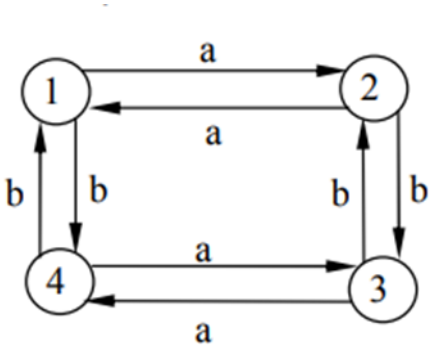
Representación de Lo Grafos Dirigidos:

Pueden usarse varias estructuras de datos para representar un dígrafo, dependiendo su selección de las operaciones que se aplicarán a los vértices y arcos del dígrafo.

Una representación común para un dígrafo G= {V, E} es la matriz de adyacencia. Suponiendo V= {1, 2,⋯}, la matriz de adyacencia de G es una matriz A de n x n de valores binarios donde A[i][j] es 1 si y sólo si hay un arco del vértice i a j y 0 si no lo hay.

Otra representación relacionada es la matriz de adyacencia etiquetada donde A[i][j] es la etiqueta del arco que va de i a j. Si tal arco no existe puede usarse una etiqueta especial para ese caso.

Ejemplo: A continuación, se muestra un dígrafo y su correspondiente matriz de adyacencia.

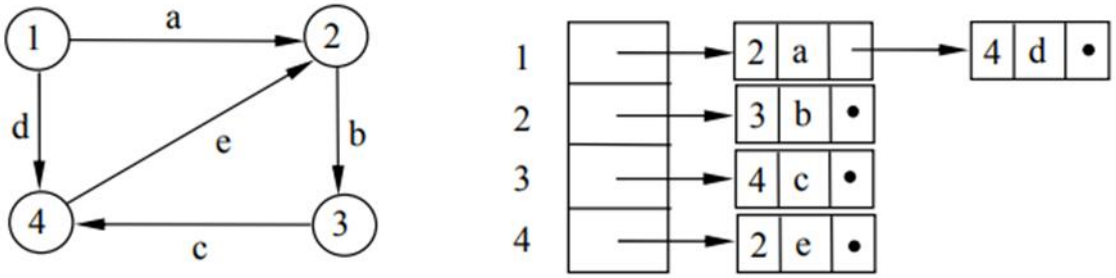


	1	2	3	4
1		a		b
2	a		b	
3		b		a
4	b		a	

La principal desventaja de usar la matriz de adyacencia es que requiere Ω(n^2) de memoria, aun cuando se tenga un dígrafo con menos de n^2 arcos. Para leer o examinar la matriz se requiere un tiempo de O(n^2).

Para evitar esta desventaja se puede usar otra representación para un dígrafo G= (V, E) llamada lista de adyacencia. La lista de adyacencia de un vértice i es una lista, en algún orden, de todos los vértices adyacentes hacia i. Se puede representar G mediante un array HEAD donde HEAD[i] es un puntero a la lista de adyacencia del vértice i.

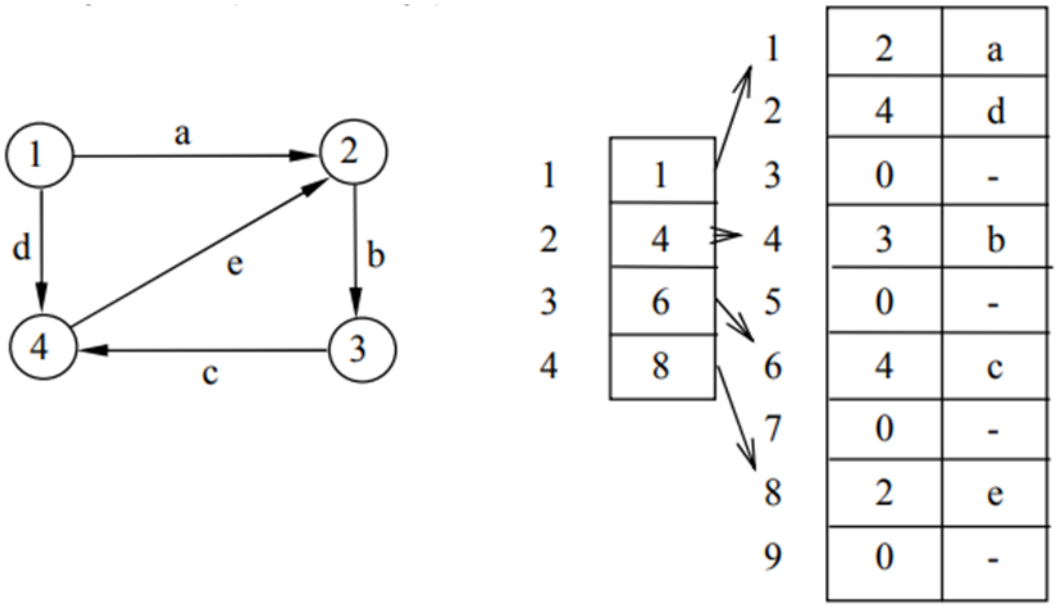
Ejemplo: A continuación, se muestra un dígrafo y su representación usando la lista de adyacencia. Se ha usado una lista enlazada simple.



La representación mediante lista de adyacencia requiere una memoria proporcional al número de vértices más el número de arcos. Se usa cuando el número de arcos es mucho menor a n^2. La desventaja es que podemos tomar un tiempo O(n) para determinar si hay un arco del vértice i a la j.

Si se espera que el grafo permanezca fijo (pocos o ningún cambio en la lista deAdyacencia) puede usarse HEAD[i] como un cursor de un array ADJ, donde ADJ [HEAD[i]] contiene los vértices adyacentes a i, hasta que se encuentre un final de lista (puede ser un 0).

Ejemplo: Para el siguiente dígrafo se muestra otra representación de la lista de adyacencia (uso de arrays).



Operaciones Básicas en Grafos

Los grafos, como todas las estructuras de datos, tienen dos operaciones fundamentales: inserción y eliminación. Estas operaciones se subdividen en insertar/eliminar vértices e insertar/eliminar aristas.

Insertar Vértice

La inserción de un nuevo vértice en un grafo es una operación simple. Consiste en agregar una nueva entrada en la estructura de datos que almacena los vértices. Una vez realizado esto, el grafo tendrá un vértice adicional, inicialmente aislado, ya que no tendrá aristas conectadas a él. En términos prácticos, esto implica añadir el nuevo vértice a una lista o conjunto de vértices.

Insertar Arista

La inserción de una nueva arista también es una operación directa. Para añadir una arista entre dos vértices, es necesario actualizar la lista de adyacencia del vértice de origen para incluir al vértice de destino. Por ejemplo, al añadir la arista (A, C), se agrega C a la lista de adyacencia de A, indicando que ahora A tiene una conexión directa con C.

Eliminar Vértice

Eliminar un vértice es el proceso inverso a su inserción. Se debe eliminar la entrada correspondiente en la tabla de vértices. Además, es necesario eliminar todas las aristas que tengan al vértice eliminado como origen o destino. Esto implica recorrer las listas de adyacencia de todos los vértices para borrar cualquier referencia al vértice que se está eliminando.

Eliminar Arista

Para eliminar una arista, es necesario eliminar el nodo destino de la lista de adyacencia del nodo origen. Esto significa que, para la arista (A, C), se debe eliminar C de la lista de adyacencia de A.

Otras Operaciones en Grafos

. Búsqueda de Elementos

Definición: La búsqueda de elementos en un grafo implica determinar si un vértice o arista específico existe en la estructura del grafo. Es fundamental para operaciones de actualización, validación y análisis del grafo.

Recorridos del Grafo

Definición: Los recorridos del grafo son métodos para visitar todos los vértices o aristas de un grafo de manera sistemática. Los dos métodos más comunes son el Recorrido en Profundidad (DFS) y el Recorrido en Amplitud (BFS).

Algoritmos de Caminos Mínimos

Definición: Estos algoritmos se utilizan para encontrar la ruta más corta entre dos vértices en un grafo ponderado. El algoritmo de Dijkstra es adecuado para grafos con pesos no negativos, mientras que el algoritmo de Bellman-Ford puede manejar pesos negativos.

Algoritmos de Conectividad

Definición: Estos algoritmos determinan cómo están conectados los vértices en un grafo. Identifican componentes conectados, puentes, puntos de articulación y verifican si el grafo es conexo.

Algoritmos de Flujo de Red

Definición: Utilizados para resolver problemas de flujo máximo en redes, estos algoritmos determinan la cantidad máxima de flujo que se puede enviar desde una fuente a un sumidero en una red de capacidad. El algoritmo de Ford-Fulkerson es uno de los más conocidos.

Detección de Ciclos

Definición: Estos algoritmos identifican si un grafo contiene ciclos, es decir, secuencias de aristas que comienzan y terminan en el mismo vértice sin repetir ninguna arista. El algoritmo de Tarjan es eficiente para detectar ciclos en grafos dirigidos.

Coloración de Grafos

Definición: La coloración de grafos es el proceso de asignar colores a los vértices de un grafo de modo que no haya dos vértices adyacentes que compartan el mismo color. Es fundamental en la teoría de grafos y tiene aplicaciones en la programación de horarios y otros problemas de optimización.

Ejemplos

Insertar Vértice

Ejemplo: Supongamos que tenemos un grafo con los vértices A y B. Si queremos añadir un nuevo vértice C, simplemente añadimos C a la lista de vértices.

- Lista de vértices antes: {A, B}
- Lista de vértices después: {A, B, C}

Insertar Arista

Ejemplo: Tenemos un grafo con los vértices A, B y C. Si queremos añadir una arista entre A y C:

- Lista de adyacencia antes: A -> {B}, B -> {A}, C -> {}
- Lista de adyacencia después: A -> {B, C}, B -> {A}, C -> {A}

Eliminar Vértice

Ejemplo: Supongamos que queremos eliminar el vértice B del grafo que tiene los vértices A, B y C, con las aristas (A-B) y (B-C):

- Lista de vértices antes: {A, B, C}
- Lista de adyacencia antes: A -> {B}, B -> {A, C}, C -> {B}
- Lista de vértices después: {A, C}

- Lista de adyacencia después: A -> {}, C -> {}

Eliminar Arista

Ejemplo: En un grafo con los vértices A, B y C y las aristas (A-B) y (A-C), si eliminamos la arista (A-C):

- Lista de adyacencia antes: A -> {B, C}, B -> {A}, C -> {A}
- Lista de adyacencia después: A -> {B}, B -> {A}, C -> {}

Búsqueda de Elementos

Ejemplo: Buscar si el vértice D está en el grafo con vértices A, B y C.

- Resultado: El vértice D no está en el grafo.

Recorridos del Grafo

Ejemplo: Utilizando DFS en un grafo con vértices A, B, C y las aristas (A-B), (A-C):

- Recorrido DFS desde A: A -> B -> C

Algoritmos de Caminos Mínimos

Ejemplo: Usando el algoritmo de Dijkstra para encontrar el camino más corto desde A hasta C en un grafo con pesos en las aristas: (A-B, 1), (B-C, 2), (A-C, 4):

- Camino más corto desde A hasta C: A -> B -> C, con peso total 3.

Algoritmos de Conectividad

Ejemplo: Determinar los componentes conectados en un grafo con vértices A, B, C y D y aristas (A-B), (C-D):

- Componentes conectados: {A, B}, {C, D}

Algoritmos de Flujo de Red

Ejemplo: Utilizando el algoritmo de Ford-Fulkerson para encontrar el flujo máximo en una red de capacidad entre los nodos S (origen) y T (destino).

- Supongamos una red con capacidades: (S-A, 10), (A-T, 5), (S-B, 10), (B-T, 5)
- Flujo máximo: 10 (S -> A -> T con flujo 5, S -> B -> T con flujo 5)

Detección de Ciclos

Ejemplo: Usando el algoritmo de Tarjan para detectar ciclos en un grafo con aristas (A-B), (B-C), (C-A):

- Resultado: Hay un ciclo en el grafo: A -> B -> C -> A

Coloración de Grafos

Ejemplo: Colorear un grafo con vértices A, B, C, D y aristas (A-B), (A-C), (B-D), (C-D):

- Coloración posible: A (color 1), B (color 2), C (color 2), D (color 1)

Video grupal de exposición:

https://drive.google.com/file/d/1-g6fEMfj2tFt4DRt2GmqHOT09cUXGBCZ/view?usp=sharing

Objetivos

Objetivo General

Comprender y aplicar los conceptos y tipos de grafos en la programación para la representación y análisis de relaciones complejas entre objetos o entidades, destacando su relevancia y utilidad en diversos contextos de sistemas de información

Objetivos Específicos

-Definir los conceptos básicos y tipos de grafos: Proporcionar una explicación clara de qué son los grafos, sus componentes (vértices y aristas) y los diferentes tipos que existen, como grafos simples, multígrafos, grafos completos, etiquetados y ponderados.

-Explicar las aplicaciones y usos de los grafos: Describir cómo los grafos son utilizados en diversos campos, incluyendo la gestión de clientes, la prevención del lavado de dinero, la recomendación de productos y la seguridad cibernética.

-Analizar las propiedades y características de los grafos: Examinar las ventajas y desventajas de los grafos, así como las propiedades que los hacen útiles para representar relaciones complejas en la programación

Video de exposición: https://drive.google.com/file/d/1-g6fEMfj2tFt4DRt2GmqHOT09cUXGBCZ/view?usp=sharing

Conclusión

Los grafos son herramientas esenciales en la programación debido a su capacidad para modelar y resolver una amplia variedad de problemas complejos. Su estructura permite representar de manera eficiente las relaciones entre diferentes entidades, lo cual es invaluable en el análisis y gestión de datos. A lo largo del estudio de los grafos, se han destacado sus tipos, aplicaciones, propiedades y operaciones fundamentales. La comprensión y el uso de grafos permiten a los programadores y científicos de datos abordar problemas desde una perspectiva más estructurada y lógica. La adopción de tecnologías basadas en grafos en diversos campos, como la seguridad cibernética, la gestión de clientes y los sistemas de recomendación, demuestra su importancia y relevancia en el mundo actual.

EGRAFÍA

- ¿Qué es un grafo en programación, para qué sirve, partes, características y tipos? Conoce el tipo abstracto de datos | Qué es (quees.com)
- https://www.apinem.com/grafos-en-programacion/#8-grafos-ponderados
- Funcion de los grafos en Programación
- Representación de Grafos Dirigidos
- https://www.hci.uniovi.es/Products/DSTool/grafos/grafos-operaciones.html