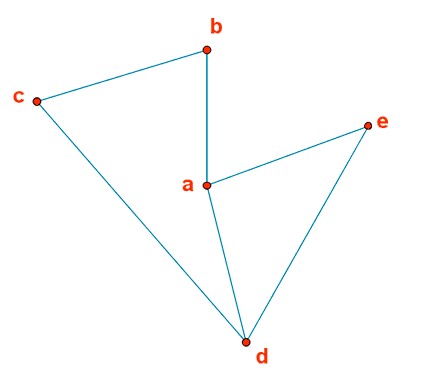
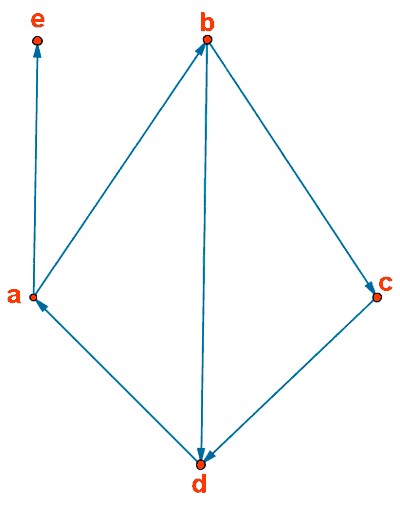
Los grafos son estructuras de datos que representan relaciones entre objetos. Estos objetos son llamados nodos o vértices, y las relaciones entre ellos son llamadas aristas. El análisis de grafos es una rama fundamental de la informática que se utiliza para resolver una gran variedad de problemas en distintos campos, desde la biología hasta la logística.

|  |  |  |
| --- | --- | --- |
| **Tipo de Grafo** | **Definición** | **Ejemplo de Uso** |
| **Grafo No**  **Dirigido** | Las aristas son bidireccionales | Redes sociales, relaciones de amistad |
| **Grafo Dirigido** | Las aristas tienen una dirección | Sistemas de navegación, flujo de datos |
| **Grafo No**  **Ponderado** | Las aristas no tienen pesos asociados | Redes de computadores, relaciones de conexión |
| **Grafo**  **Ponderado** | Las aristas tienen pesos asociados | Mapas de carreteras, rutas de vuelo |
| **Matriz de**  **Adyacencia** | Representación de grafo como matriz bidimensional | Determinar conexiones y pesos rápidamente |
| **Listas de**  **Adyacencia** | Representación de grafo mediante listas de vecinos | Algoritmos de búsqueda y recorrido de grafos |

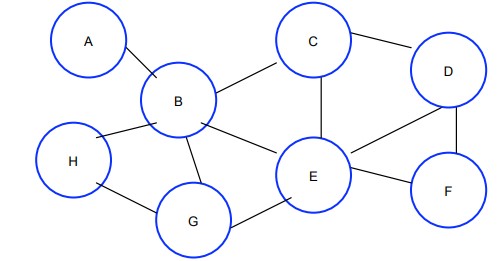
1. **Grafos dirigidos y no dirigidos:**
   * Grafo no dirigido: En este tipo de grafo, las aristas son bidireccionales y no tienen una dirección específica. Es decir, la relación entre dos nodos es simétrica. Si hay una arista que conecta el nodo A con el nodo B, también hay una arista que conecta el nodo B con el nodo A. Un ejemplo común de grafo no dirigido es una red social, donde los nodos son personas y las aristas representan la amistad entre ellas.



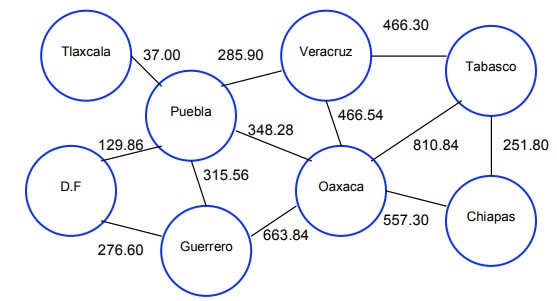
* + Grafo dirigido: En este tipo de grafo, las aristas tienen una dirección específica. La relación entre dos nodos puede ser unidireccional. Si hay una arista que va desde el nodo A al nodo B, no necesariamente habrá una arista que vaya en la dirección opuesta. Un ejemplo de grafo dirigido es un sistema de navegación, donde los nodos representan ubicaciones y las aristas indican las direcciones permitidas entre ellas.



1. **Grafos ponderados y no ponderados:** 
   * Grafo no ponderado: En este tipo de grafo, las aristas no tienen ningún peso o valor asociado. Simplemente representan la conexión entre dos nodos. Un ejemplo común es un grafo que representa las conexiones de una red de computadores, donde las aristas indican si los computadores están conectados o no.

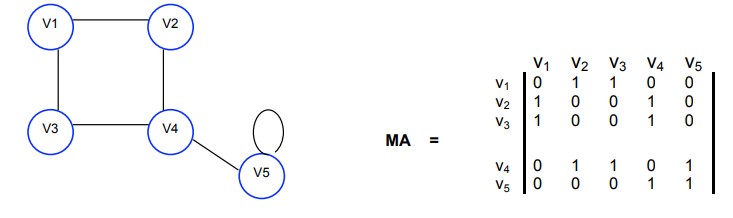


* + Grafo ponderado: En este tipo de grafo, cada arista tiene un peso o valor asociado. El peso puede representar una medida de distancia, costo, tiempo u otro criterio relevante en el contexto del problema. Un ejemplo de grafo ponderado es un mapa de carreteras, donde las aristas representan las carreteras y el peso indica la distancia entre dos ciudades.



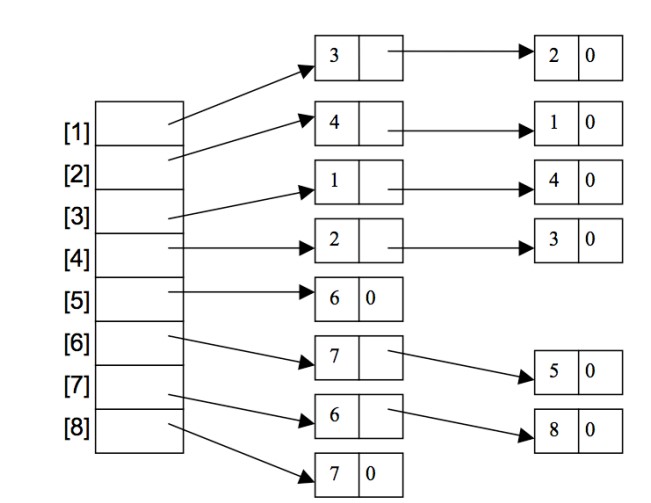
1. **Matriz de adyacencia:**

La matriz de adyacencia es una representación de un grafo utilizando una matriz bidimensional. Cada fila y columna de la matriz representa un nodo y el valor en la posición (i, j) indica si hay una arista que conecta los nodos i y j. En el caso de un grafo ponderado, los valores en la matriz pueden representar los pesos de las aristas. La matriz de adyacencia es útil para determinar rápidamente si existe una conexión directa entre dos nodos y también para consultar los pesos de las aristas. Sin embargo, puede requerir más espacio de almacenamiento si el grafo es grande y disperso.



1. **Listas de adyacencia:**

Las listas de adyacencia representan un grafo utilizando una estructura de datos en la que cada nodo tiene una lista de sus nodos vecinos. Cada nodo en la lista está asociado con una arista. En el caso de un grafo ponderado, la lista puede contener los pesos de las aristas también. Las listas de adyacencia son eficientes en términos de espacio de almacenamiento, especialmente para grafos dispersos. También facilitan la búsqueda de los vecinos de un nodo y son útiles en algoritmos de búsqueda y recorrido de grafos.



**Ejemplos de uso:**

* + Un grafo no dirigido se puede utilizar para modelar las conexiones de amigos en una red social.
  + Un grafo dirigido ponderado se puede utilizar para representar las rutas de vuelo entre diferentes ciudades, donde los pesos de las aristas indican la distancia o el costo del vuelo.
  + La matriz de adyacencia se puede utilizar para determinar rápidamente si dos nodos están conectados directamente en un grafo y para consultar los pesos de las aristas en un grafo ponderado.
  + Las listas de adyacencia son útiles en algoritmos como BFS o DFS, donde necesitas acceder a los nodos vecinos de cada nodo de manera eficiente.

Dentro del análisis de grafos, los algoritmos de búsqueda son muy importantes. Los algoritmos de búsqueda permiten encontrar caminos entre nodos en un grafo, lo que tiene aplicaciones muy diversas. Entre los algoritmos de búsqueda más comunes se encuentran DFS (Depth-First Search) y BFS (Breadth-First Search).

DFS trabaja recursivamente y va “profundizando” en un camino del grafo antes de avanzar a otros caminos. BFS, en cambio, examina todos los nodos a una distancia dada antes de avanzar a los nodos a una distancia mayor.

Ambos algoritmos tienen sus fortalezas y debilidades dependiendo de la tarea que se quiera realizar. DFS es útil cuando se busca una solución en profundidad, lo que puede ser útil en problemas que tengan múltiples soluciones en diferentes niveles del grafo. Por otro lado, BFS es ideal para encontrar la solución más corta o una solución cercana al nodo objetivo.

Por otro lado, el algoritmo DFS (Búsqueda en Profundidad) se enfoca en explorar o buscar en el grafo hacia abajo en profundidad antes de retroceder. A diferencia de BFS, DFS no sigue un enfoque de capas, sino que sigue un camino específico hasta llegar a un nodo sin nodos vecinos no visitados. Luego, retrocede al último nodo con nodos vecinos no visitados y continúa explorando desde allí. El proceso general del algoritmo DFS es el siguiente:

1. Comienza por un nodo inicial y márcalo como visitado.
2. Explora un nodo vecino no visitado.
3. Repite el paso 2 para el nodo vecino no visitado más cercano.
4. Si no hay más nodos vecinos no visitados, retrocede al nodo anterior y continúa la exploración desde allí.

DFS puede sumergirse rápidamente en una rama específica del grafo, explorando en profundidad antes de retroceder. Esto lo hace útil para problemas como encontrar componentes conectados, detectar ciclos en un grafo, recorrer todos los nodos de un grafo y problemas de backtracking.

# BFS (Breadth-First Search, búsqueda en anchura)

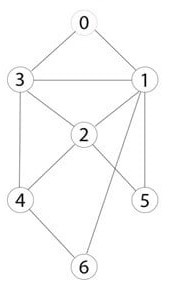
Su enfoque principal es explorar el grafo en capas o niveles, comenzando desde un nodo inicial y avanzando hacia los nodos vecinos antes de pasar a los nodos más distantes.

BFS garantiza que se visiten primero los nodos más cercanos al nodo inicial antes de avanzar a los nodos más lejanos. Esto hace que BFS sea especialmente útil para encontrar el camino más corto entre dos nodos en un grafo ponderado o no ponderado, ya que explora sistemáticamente todas las rutas posibles antes de avanzar a las siguientes.

Este algoritmo selecciona un solo nodo (punto inicial o fuente) en un gráfico y luego visita todos los nodos adyacentes al nodo seleccionado. Una vez que el algoritmo visita y marca el nodo de inicio, se mueve hacia los nodos no visitados más cercanos y los analiza.

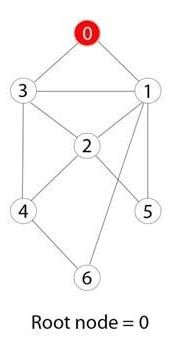
**Ejemplo de BFS**

Paso 1)



Tiene una gráfica de siete números que van del 0 al 6.

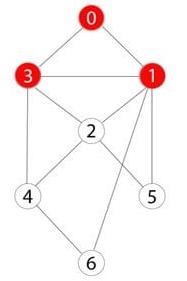
Paso 2)



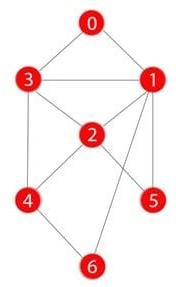
Se ha marcado 0 o cero como nodo raíz.

0 es visitado, marcado.

Paso 3)



Paso 4)



El desplazamiento se repite hasta se visitan todos los nodos.

# DFS (Depth-First Search, búsqueda en profundidad)

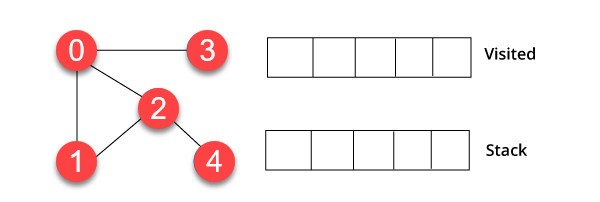
El algoritmo DFS (Búsqueda en Profundidad) se enfoca en explorar o buscar en el grafo en profundidad antes de retroceder. DFS sigue un camino específico hasta llegar a un nodo sin nodos vecinos no visitados. Luego, retrocede al último nodo con nodos vecinos no visitados y continúa explorando desde allí. El proceso general del algoritmo DFS es el siguiente:

1. Comienza por un nodo inicial y márcalo como visitado.
2. Explora un nodo vecino no visitado.
3. Repite el paso 2 para el nodo vecino no visitado más cercano.
4. Si no hay más nodos vecinos no visitados, retrocede al nodo anterior y continúa la exploración desde allí.

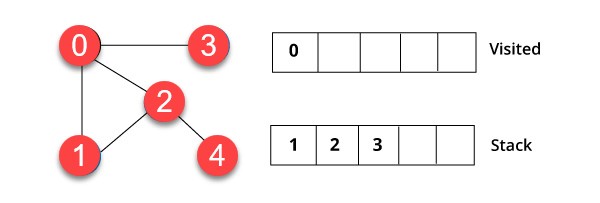
DFS es un algoritmo para encontrar o recorrer gráficos o árboles en dirección de profundidad. La ejecución del algoritmo comienza en el nodo raíz y explora cada rama antes de retroceder.

**Ejemplo de DFS**

En el siguiente ejemplo de DFS, hemos utilizado un gráfico no dirigido que tiene 5 vértices.

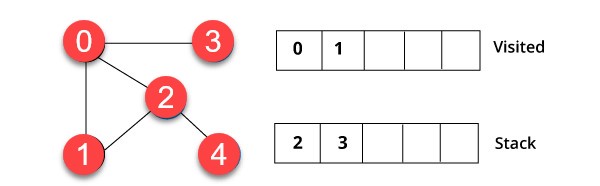


Paso 1)

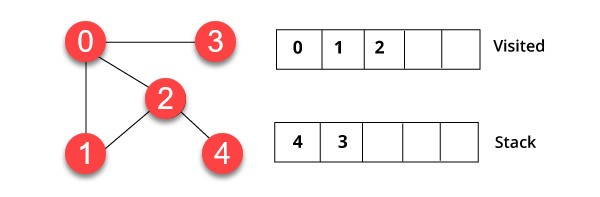


Comienza desde el vértice 0. El algoritmo comienza poniéndolo en la lista de visitados y colocando simultáneamente todos sus vértices adyacentes en los datos estructura llamada pila.

Paso 2)

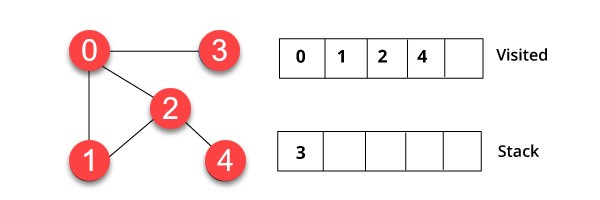


Visitará el elemento , que está en la parte superior de la pila, por ejemplo, 1 y vaya a sus nodos adyacentes. Es porque ya se ha visitado 0. Por lo tanto, visita el vértice 2. Paso 3)

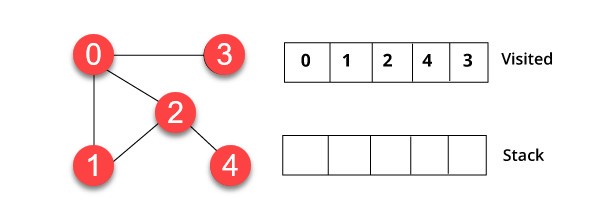


El vértice 2 tiene un vértice cercano no visitado en 4. Por lo tanto, lo agrega en la pila y lo visita.

Paso 4)



Finalmente, visita el último vértice 3, no tiene ningún nodo contiguo no visitado. Ha completado el recorrido del grafo mediante el algoritmo DFS.



Algunos ejemplos de uso principal de cada algoritmo:

Ejemplo de uso principal de BFS:

En un mapa de carreteras, BFS se puede utilizar para encontrar el camino más corto entre dos ciudades. Cada ciudad se representa como un nodo y las carreteras como aristas entre los nodos correspondientes

BFS explorará sistemáticamente las ciudades más cercanas al inicio antes de avanzar hacia las más lejanas, encontrando así el camino más corto.

Ejemplo de uso principal de DFS:

En un laberinto, DFS se puede utilizar para encontrar una solución. Cada cruce o intersección en el laberinto se representa como un nodo y los pasillos como aristas entre los nodos correspondientes. DFS seguirá un camino específico hasta que se encuentre con un punto sin salidas y luego retrocederá para explorar otras opciones hasta encontrar la salida.

En resumen, BFS y DFS son algoritmos de búsqueda en grafos que se utilizan para explorar y buscar información en estructuras de datos de grafos.

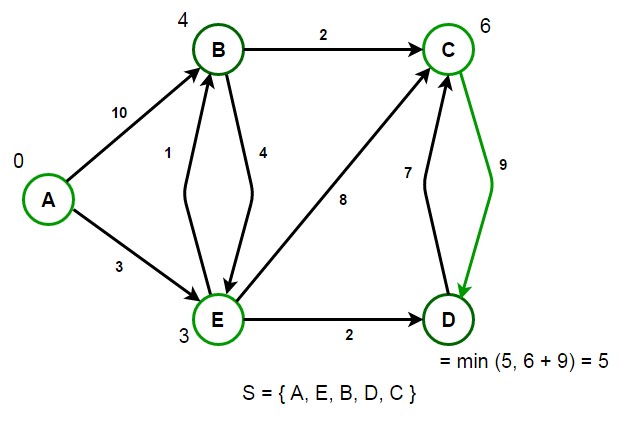
La principal diferencia radica en el enfoque de exploración: BFS explora en capas o niveles, mientras que DFS sigue un camino específico en profundidad antes de retroceder.

# Algoritmo de Dijkstra

El algoritmo de Dijkstra se utiliza para encontrar el camino más corto desde un nodo inicial a todos los demás nodos en un grafo ponderado no dirigido o dirigido, donde cada arista tiene asociado un peso o costo. Este algoritmo resuelve el problema conocido como el problema del camino más corto.

Como funciona el algoritmo de Dijkstra:

El algoritmo de Dijkstra garantiza que, en cada paso, se seleccione el nodo con la distancia actual mínima y se actualicen las distancias de sus nodos vecinos si se encuentra un camino más corto. Una vez que todos los nodos hayan sido visitados, se habrá encontrado la distancia más corta desde el nodo inicial a todos los demás nodos.



Las principales diferencias entre el algoritmo de Dijkstra y otros algoritmos de búsqueda en grafos, como BFS o DFS, son:

1. Dijkstra encuentra el camino más corto desde un nodo inicial hacia todos los demás nodos, mientras que BFS encuentra el camino más corto desde un nodo inicial hacia un nodo objetivo específico.
2. Dijkstra funciona en grafos ponderados, es decir, donde las aristas tienen pesos asociados, mientras que BFS y DFS no tienen en cuenta los pesos de las aristas.
3. Dijkstra utiliza una estrategia de "selección del nodo con la distancia actual mínima", en lugar de un enfoque de exploración en capas o en profundidad.

Un ejemplo común de uso del algoritmo de Dijkstra es en la planificación de rutas de transporte. Puede utilizarse para encontrar la ruta más corta y eficiente para llegar desde un punto A a un punto B en una red de carreteras o en una red de transporte público, considerando los diferentes costos asociados con cada arista, como la distancia, el tiempo o el costo monetario.

En resumen, el algoritmo de Dijkstra es un algoritmo de búsqueda en grafos que encuentra el camino más corto desde un nodo inicial a todos los demás nodos en un grafo ponderado. Su principal diferencia radica en su enfoque de encontrar la distancia mínima en cada paso y su capacidad para manejar grafos con pesos asociados a las aristas. Es ampliamente utilizado en problemas de planificación de rutas y optimización en redes de transporte.