Video35 (1)

Conceptos Principales

1. a
2. b
3. c

Notas**Resumen general**

Este video introduce cómo **representar grafos en código** y qué **funciones básicas** debe tener una estructura de datos de tipo grafo. También explica el concepto de **adyacencia** y cómo determinar si dos nodos están conectados.

---

1. ¿Qué es un grafo?

Un **grafo** es una estructura que contiene:

- Un conjunto de **vértices (nodos)**.
- Un conjunto de **aristas (enlaces)** que conectan esos vértices.

Los grafos pueden ser:

- **Dirigidos o no dirigidos.**
- **Valorados (con pesos)** o **no valorados**.

---

2. Representación de un grafo

Hay dos formas comunes de representarlos en programación:

Matriz de adyacencia

- Es una **matriz cuadrada** donde cada posición `(i, j)` indica si existe una arista entre los vértices `i` y `j`.
- En un grafo **no valorado**:
  - `1` → existe conexión
  - `0` → no existe
- En un grafo **valorado**:
  - Se coloca el **peso** de la arista (por ejemplo, distancia o costo).

Lista de adyacencia

- Cada vértice tiene una **lista** con los nodos a los que está conectado.
- Es más eficiente en memoria cuando el grafo tiene **pocas conexiones**.

---

3. Funciones básicas de un grafo

Toda implementación de grafos debería contar con estas funciones (aunque algunas son opcionales dependiendo del uso):

Función	Descripción	Parámetros / Retorno
`agregarArista(i, j)`	Crea una conexión entre dos vértices.	`(i, j)` y, si es valorado, un `peso`.
`eliminarArista(i, j)`	Quita una conexión existente.	`(i, j)`
`agregarVertice()`	Añade un nuevo vértice al grafo.	Modifica la matriz/lista.
`eliminarVertice(v)`	Elimina un vértice y sus aristas.	`(v)`
`esAdyacente(i, j)`	Verifica si hay conexión entre dos vértices.	Retorna `True` o `False`.
`retornarAdyacentes(v)`	Devuelve todos los nodos conectados a `v`.	Retorna lista o vector.

Estas funciones permiten manipular la estructura antes de aplicar algoritmos más avanzados como BFS, DFS, Dijkstra, etc.

---

4. Concepto de adyacencia

Definición

Dos vértices son **adyacentes** si están conectados directamente por una arista.

Ejemplo:

Si el nodo `H` está conectado con `L` y `K`, entonces los **adyacentes de H** son `{L, K}`.

Cómo determinarlo (en matriz de adyacencia)

1. Identificar la **fila** que corresponde al nodo `H`.
2. Leer todos los valores en esa fila.
3. Cada columna con un valor distinto de 0 o 1 indica conexión.

En código:

```
esAdyacente(i, j):
    si matriz[i][j] == 1:
        return True
    else:
        return False
```

Para listar los adyacentes:

```
retornarAdyacentes(i):
    lista = []
    para cada columna j en matriz[i]:
        si matriz[i][j] != 0:
            lista.agregar(j)
    return lista
```

---

5. Conclusión

- Los **grafos** son estructuras versátiles que permiten modelar conexiones (como mapas, redes o relaciones).
- Se pueden representar con **matriz o lista de adyacencia**, dependiendo de la necesidad.
- Las funciones básicas como `agregar`, `eliminar`, `esAdyacente` y `retornarAdyacentes` son esenciales para trabajar con grafos.
- El **concepto de adyacencia** es la base para algoritmos de recorrido (BFS, DFS) que se verán después.