Estructura de Datos

Mg. Cinthia J. Calderon Aquiño

Semana 12

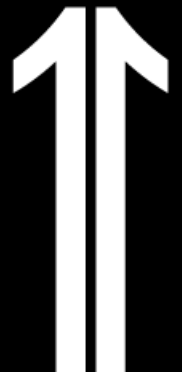
UPN.EDU.PE

Semana 12

Algoritmos para grafos: Definiciones, grafos y grafos dirigidos, aplicaciones, representación, matriz de adyacencia, lista de adyacencia, matriz de costos.

Recorrido: en amplitud (BFS), en profundidad (DFS), ordenamiento topológico y conectividad.

PRESENTACIÓN DE LA SESIÓN
Logro de la Sesión y Temario



Al término de la sesión, el estudiante desarrolla un proyecto aplicando estructuras de datos dinámicas en el lenguaje C# con entorno gráfico; demostrando capacidad de análisis, pensamiento lógico y buenas prácticas de programación e implementa funciones definidas de usuarios y sus diferentes tipos, Utilizando Grafos

- Algoritmos para grafos: Definiciones, grafos y grafos dirigidos, aplicaciones, representación, matriz de adyacencia, lista de adyacencia, matriz de costos
- Recorrido: en amplitud (BFS), en profundidad (DFS), ordenamiento topológico y conectividad.

GRAFO

Un **GRAFO** es una estructura de datos matemática que consta de un conjunto de vértices (también llamados nodos) y un conjunto de aristas (también llamadas bordes) que conectan estos vértices.

Existen varias formas de codificar un GRAFO. Pero, conceptualmente, **un grafo está formado por una serie de nodos.**

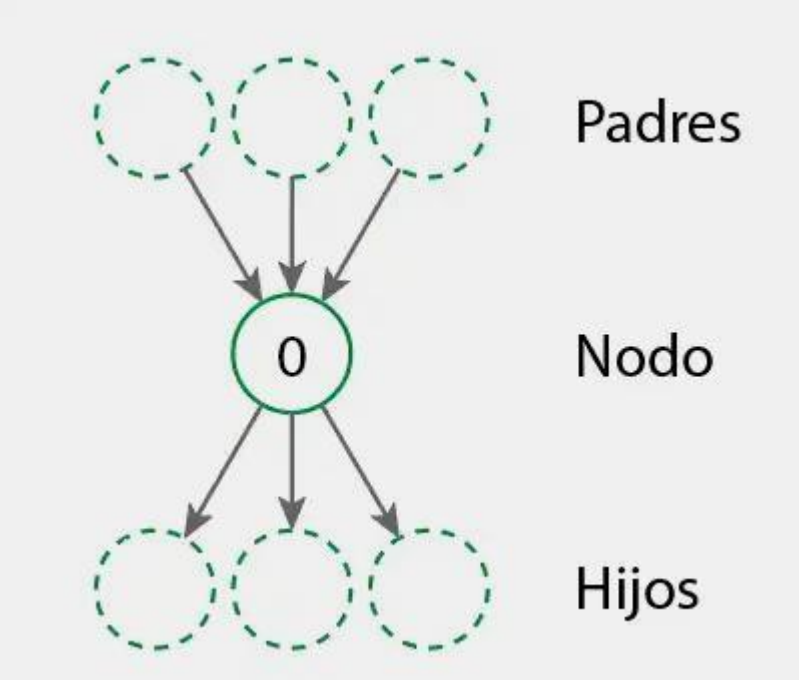
Cada uno de los nodos:

- Tiene “dentro” sus propios datos
- Tiene relaciones con otros nodos

Las relaciones entre nodos **se realizan entre mediante referencias**. Juntos, esa maraña de nodos y relaciones, forman el **GRAFO** en su conjunto.

COMPONENTES DE UN GRAFO

Nodos (vértices): Son elementos individuales en el grafo. Cada vértice puede representar entidades como personas, lugares, o cualquier objeto que estemos modelando.



Un nodo de un grafo

COMPONENTES DE UN GRAFO

Relaciones (aristas): Son las conexiones entre los vértices. Estas aristas pueden ser direccionales o no direccionales, dependiendo del tipo de relación que queremos representar.

Nodo adyacente: A los nodos que comparten una relación con el nodo actual se les denomina nodos adyacentes.

Padres e hijos: Cunando los nodos son orientados, a los nodos con relaciones entrantes y salientes se les denomina padres, o hijos.

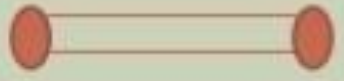
GRAFO

- **Vértices:** Son los objetos representados por punto dentro del grafo

- **Aristas:** son las líneas que unen dos vértices

- **Aristas Adyacentes:** dos aristas son adyacentes si convergen sobre el mismo vértice

- **Aristas Múltiples o Paralelas:** dos aristas son múltiples o paralelas si tienen los mismos vértices en común o incidente sobre los mismos vértices

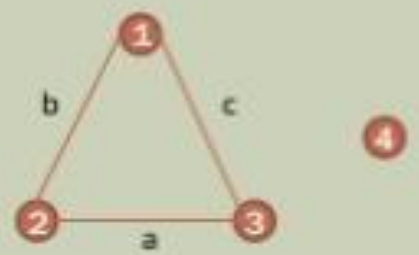


- **Lazo:** es una arista cuyos extremos inciden sobre el mismo vértice

GRAFO

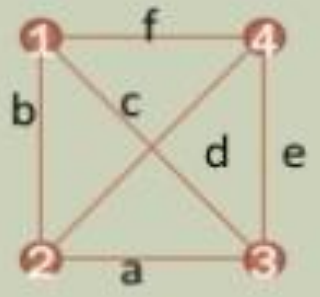
- **Arista Incidente :** Una arista es incidente a un vértice si ésta lo une a otro vértice.

- **Vértice Aislado:** Es un vértice de grado cero



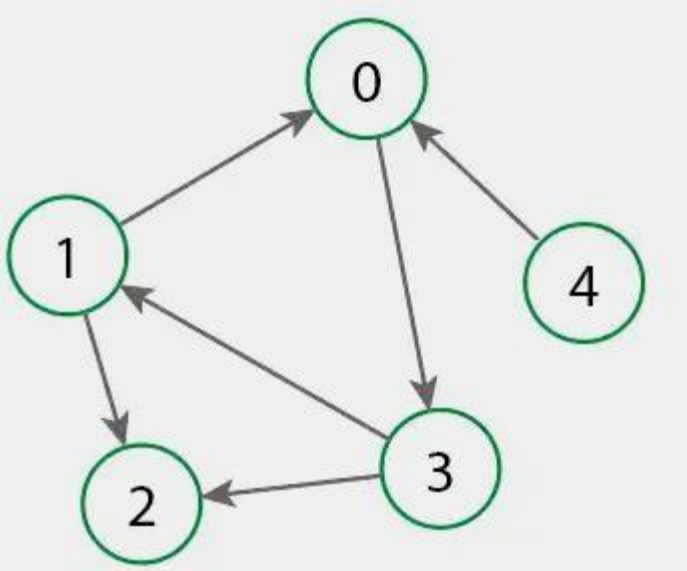
- **Vértice Pendiente:** Es aquel grafo que contiene sólo una arista, es decir tiene grado 1

- **Cruce:** Son intersecciones de las aristas en puntos diferentes a los vértices



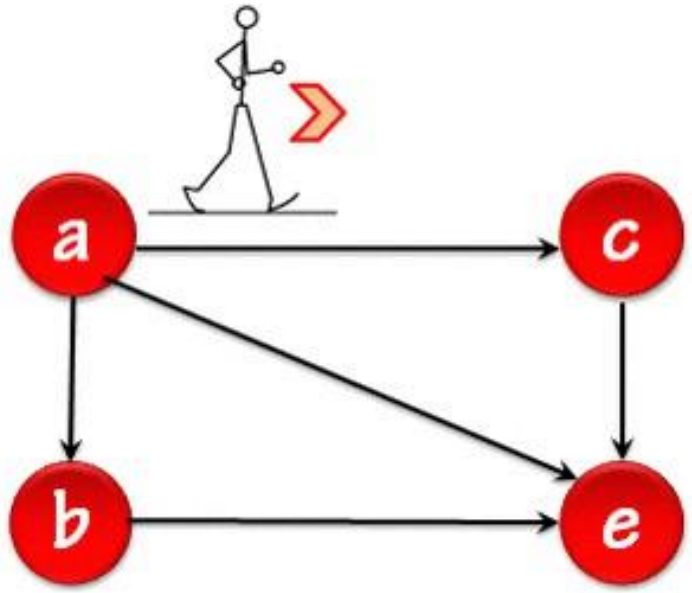
GRAFO DIRIGIDO

- **Grafo Dirigido:** En un grafo dirigido, las aristas tienen dirección. La relación entre dos vértices puede ser unidireccional. Por ejemplo, en un grafo de rutas, una carretera puede ir de A a B pero no necesariamente de B a A



GRAFO DIRIGIDO

- Un grafo dirigido es aquel en el que todas sus aristas tienen sentido o dirección.
- La relación sobre V no es simétrica. Las aristas se representan como un par ordenado (u,v).

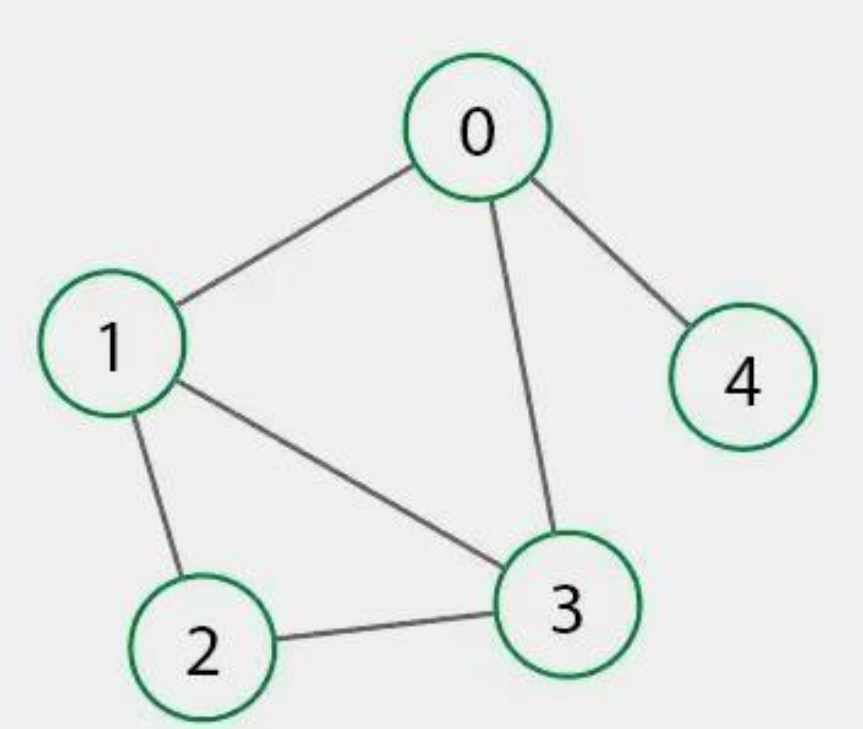


Ejemplo

- V = {a, b, c, e}
- E = {(a, b), (a, c), (a, e), (b,e), (c, e) }

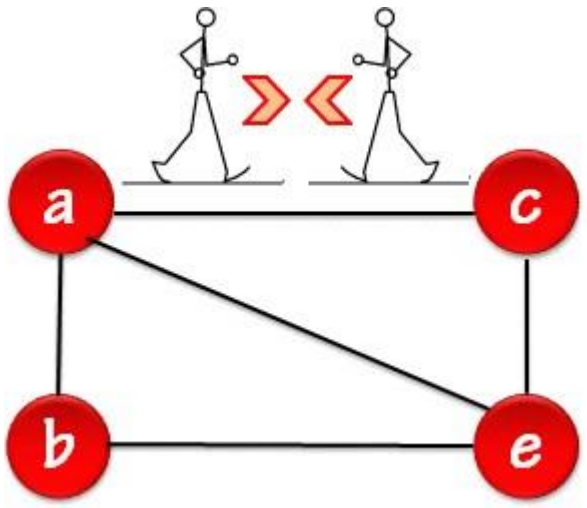
GRAFO NO DIRIGIDO

- **Grafo No Dirigido:** En este tipo de grafo, las aristas no tienen dirección. La relación entre dos vértices es bidireccional. Si A está conectado con B, entonces B también está conectado con A.



GRAFO NO DIRIGIDO

- Un grafo no dirigido es aquel en el que todas sus aristas son bidireccionales.
- La relación sobre V es simétrica. Las aristas se representan como pares no ordenados {u,v}, u,v ∈ V y u ≠ v.

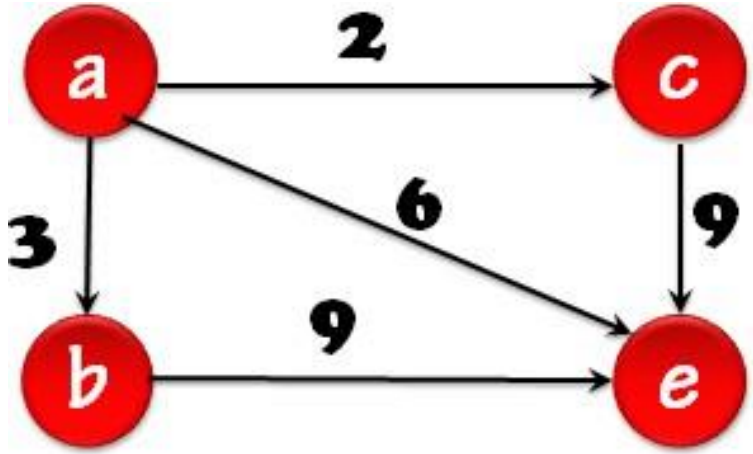


Ejemplo

V = {a, b, c, e}
E = {{a, b}, {a, c}, {a, e}, {b, e}, {c, e}}

GRAFO PONDERADO

- Un grafo ponderado, pesado o con costos es un grafo donde cada arista tiene asociado un valor o etiqueta, para representar el costo, peso, longitud, etc



GRAFO PONDERADO

- **Camino desde u Є V a v Є V:** secuencia v1, v2, ... , vk tal que u=v1, v=v k, y (vi−1,vi) Є E, para i = 2, ... ,k.
- Ej: camino desde a hasta d → <a,b,e,c,d>.

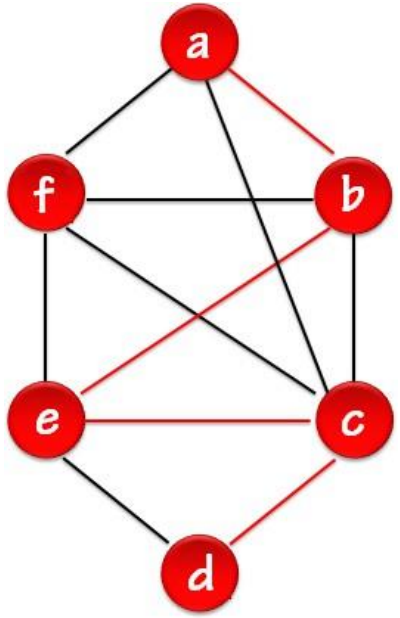


Figura a

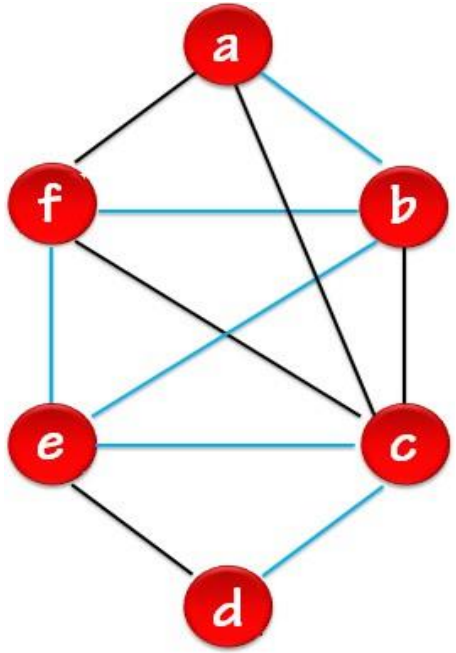


Figura b

GRAFO PONDERADO

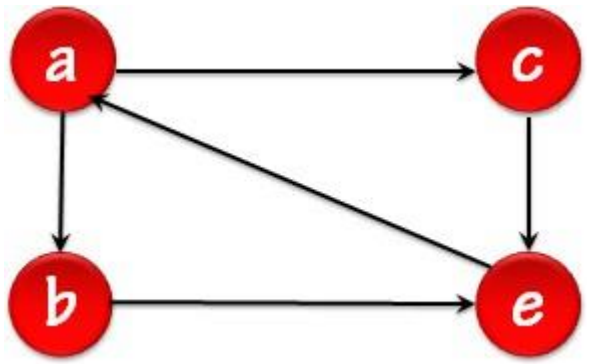
Longitud de un camino: es el número de arcos del camino.

Ejemplos:

longitud del camino desde a hasta d → <a,b,e,c,d> es 4. (figura a)

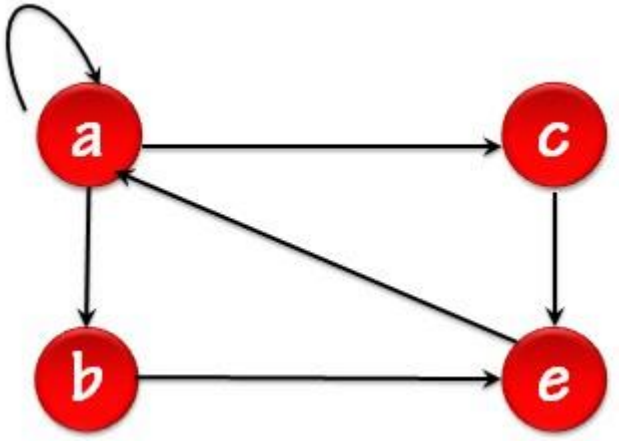
longitud del camino desde a hasta d → <a,b,e,f,b,e,c,d> es 7. (figura b)

Ciclo: Un ciclo es un camino donde el origen del camino es igual a su destino. Formalmente es camino desde v1, v2, ... , vk tal que v1=vk



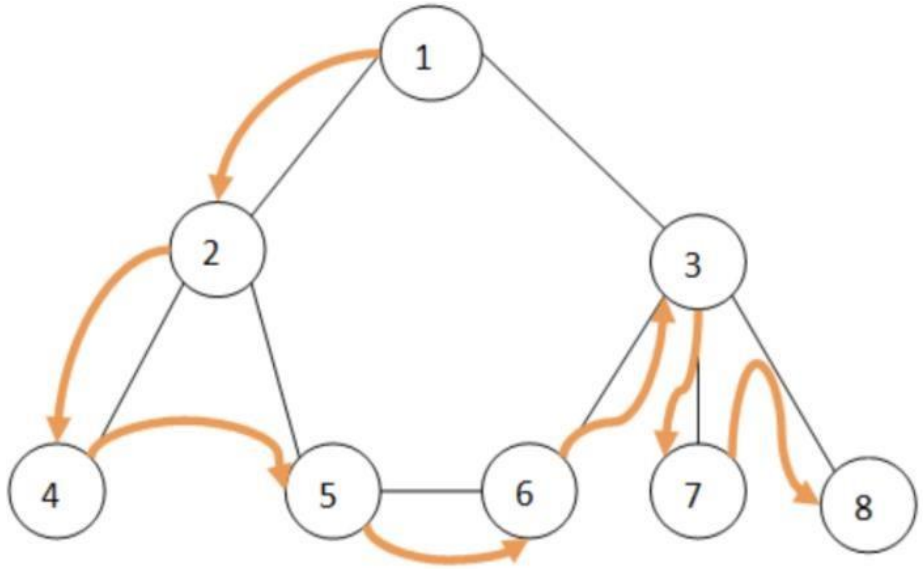
<a,c,e,a> es un ciclo de longitud 3.

Bucle: Un bucle es una arista que conecta a un vértice consigo mismo. Es un ciclo de longitud 1



BÚSQUEDA EN PROFUNDIDAD (DFS)

- Es un algoritmo de búsqueda para lo cual recorre los nodos de un grafo. Su funcionamiento consiste en ir expandiendo cada uno de los nodos que va localizando, de forma recurrente (desde el nodo padre hacia el nodo hijo). Cuando ya no quedan más nodos que visitar en dicho camino, regresa al nodo predecesor, de modo que repite el mismo proceso con cada uno de los vecinos del nodo. Cabe resaltar que si se encuentra el nodo antes de recorrer todos los nodos, concluye la búsqueda.



BÚSQUEDA EN PROFUNDIDAD (DFS)

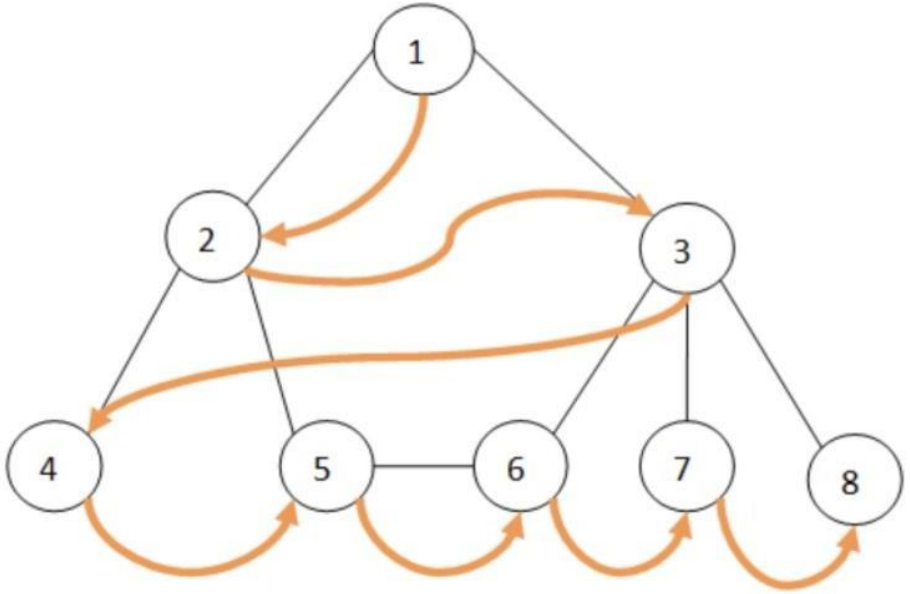
El algoritmo de búsqueda en profundidad en C#, donde se recorre los nodos del grafo usando un “iterador” que comienza desde el primer nodo hasta el último nodo ingresado. Se utiliza dos variables “nodoVisitado” y “nodoProcesado” para organizar el recorrido en profundidad, por esta razón se hace uso de la recursividad recorriendo los nodos en profundidad.

Ejemplo: Es como explorar una cueva: sigues un túnel hasta el final antes de regresar y tomar otro camino.

```
3 referencias
public void DFS(string inicio)
{
    bool[] visitado = new bool[grafo.cantVertices];
    string[] recorrido = new string[grafo.cantVertices];
    int tope = -1, cantRecorrido = 0;

    Vertice[] pila = new Vertice[grafo.cantVertices];

    int posInicio = grafo.buscar(inicio);
    if (posInicio == -1)
        return;

    pila[++tope] = grafo.vertices[posInicio];

    while (tope >= 0)
    {
        Vertice actual = pila[tope--];
        int posActual = grafo.buscar(actual.dato);

        if (!visitado[posActual])
        {
            visitado[posActual] = true;
            recorrido[cantRecorrido++] = actual.dato;

            // Agregamos vecinos en orden inverso para simular pila
            for (int i = actual.cantAristas - 1; i >= 0; i--)
            {
                Vertice vecino = actual.aristas[i].destino;
                int posVecino = grafo.buscar(vecino.dato);

                if (!visitado[posVecino])
                {
                    pila[++tope] = vecino;
                }
            }
        }
    }

    dgvRecorrido.Rows.Clear();
    for (int i = 0; i < cantRecorrido; i++)
    {
        dgvRecorrido.Rows.Add(recorrido[i]);
    }
}
```

BÚSQUEDA EN PROFUNDIDAD (BFS)

Es un algoritmo de búsqueda para lo cual recorre los nodos de un grafo, comenzando en la raíz (eligiendo algún nodo como elemento raíz en el caso de un grafo), para luego explorar todos los vecinos de este nodo. A continuación, para cada uno de los vecinos se exploran sus respectivos vecinos adyacentes, y así hasta que se recorra todo el grafo. Cabe resaltar que si se encuentra el nodo antes de recorrer todos los nodos, concluye la búsqueda.



BÚSQUEDA EN PROFUNDIDAD(BFS)

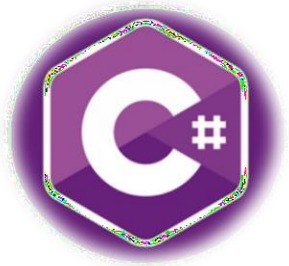
El algoritmo de búsqueda en anchura en C#, donde se recorren los nodos del grafo usando un “iterador” que comienza desde el primer nodo ingresado hasta el último nodo ingresado, recorriendo los nodos vecinos primeramente antes de los nodos hijos. Se utilizan dos variables “nodoVisitado” y “nodoProcesado” para organizar el recorrido en anchura. Por esta razón se hace uso de la estructura de datos cola (en inglés ‘queue’) para visitar los nodos en orden de llegada. Es decir, el primero procesa los nodos que primero llegaron a la cola.

Ejemplo: Es como buscar a alguien en una red social comenzando con tus amigos, luego los amigos de tus amigos, y así hasta encontrarlo

```
public void BFS(string inicio)
{
    bool[] visitado = new bool[grafo.cantVertices];
    string[] recorrido = new string[grafo.cantVertices];
    int frente = 0, final = 0;
    int cantRecorrido = 0;

    int posInicio = grafo.buscar(inicio);
    if (posInicio == -1)
        return;

    Vertice[] cola = new Vertice[grafo.cantVertices];
    cola[final++] = grafo.vertices[posInicio];
    visitado[posInicio] = true;

    while (frente < final)
    {
        Vertice actual = cola[frente++];
        recorrido[cantRecorrido++] = actual.dato;

        for (int i = 0; i < actual.cantAristas; i++)
        {
            Vertice vecino = actual.aristas[i].destino;
            int posVecino = grafo.buscar(vecino.dato);

            if (!visitado[posVecino])
            {
                cola[final++] = vecino;
                visitado[posVecino] = true;
            }
        }
    }

    dgvRecorrido.Rows.Clear();
    for (int i = 0; i < cantRecorrido; i++)
    {
        dgvRecorrido.Rows.Add(recorrido[i]);
    }
}
```

CARACTERISTICAS

Caracteristica	BFS	DFS
Estructura usada	Cola (queue)	Pila (stack) o recursión
Orden de recorrido	Por niveles (anchura)	Por profundidad
Ideal para...	Buscar el camino más corto	Explorar estructuras completas
Memoria	Puede usar más memoria	Usa menos memoria (en muchos casos)

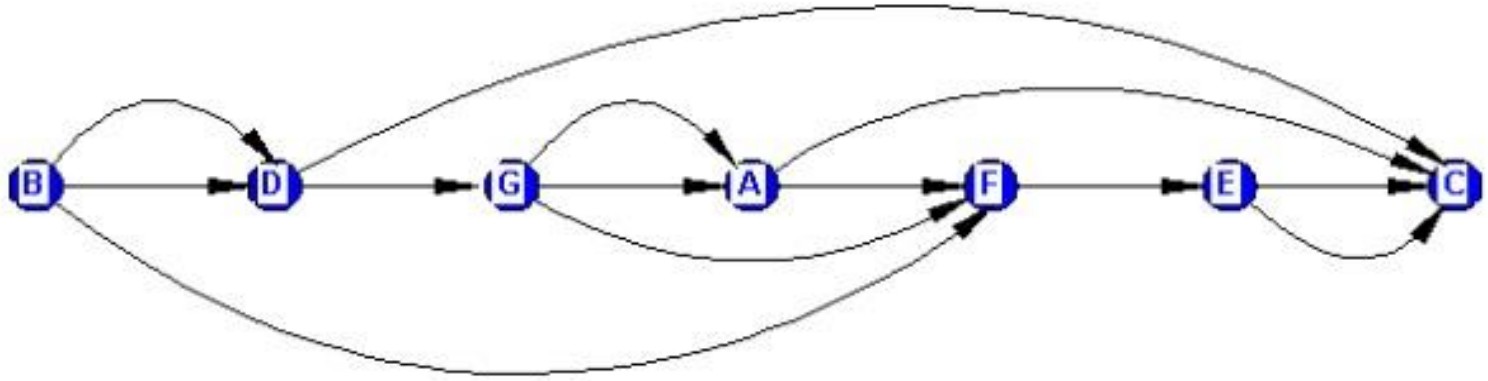


ORDENAMIENTO TOPOLÓGICO

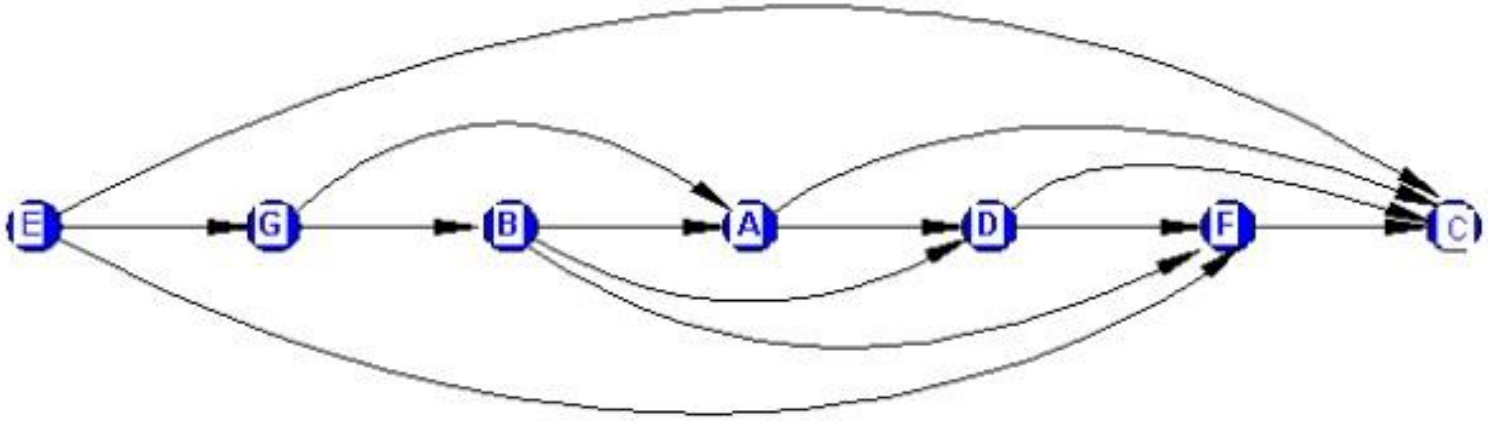
Una forma simple de pensar en el Ordenamiento topológico es con el siguiente problema: En una fábrica existen n diferentes tareas que se tienen que llevar a cabo, sin embargo, hay algunas tareas que tienen una serie de tareas requisito que se tienen que completar antes de poder ejecutarlas. Las tareas están numeradas del 1 al n, y al jefe de la fábrica le interesa que se les dé prioridad a las tareas que son más importantes, (la tarea i es más importante que la tarea j si i>j). Si se conocen los requisitos para cada tarea, da algún orden en el que se pueden llevar a cabo las tareas respetando los requisitos de cada una y la preferencia del jefe.

https://arrobaricardoge.github.io/Kahn-TopoSort-Visual/

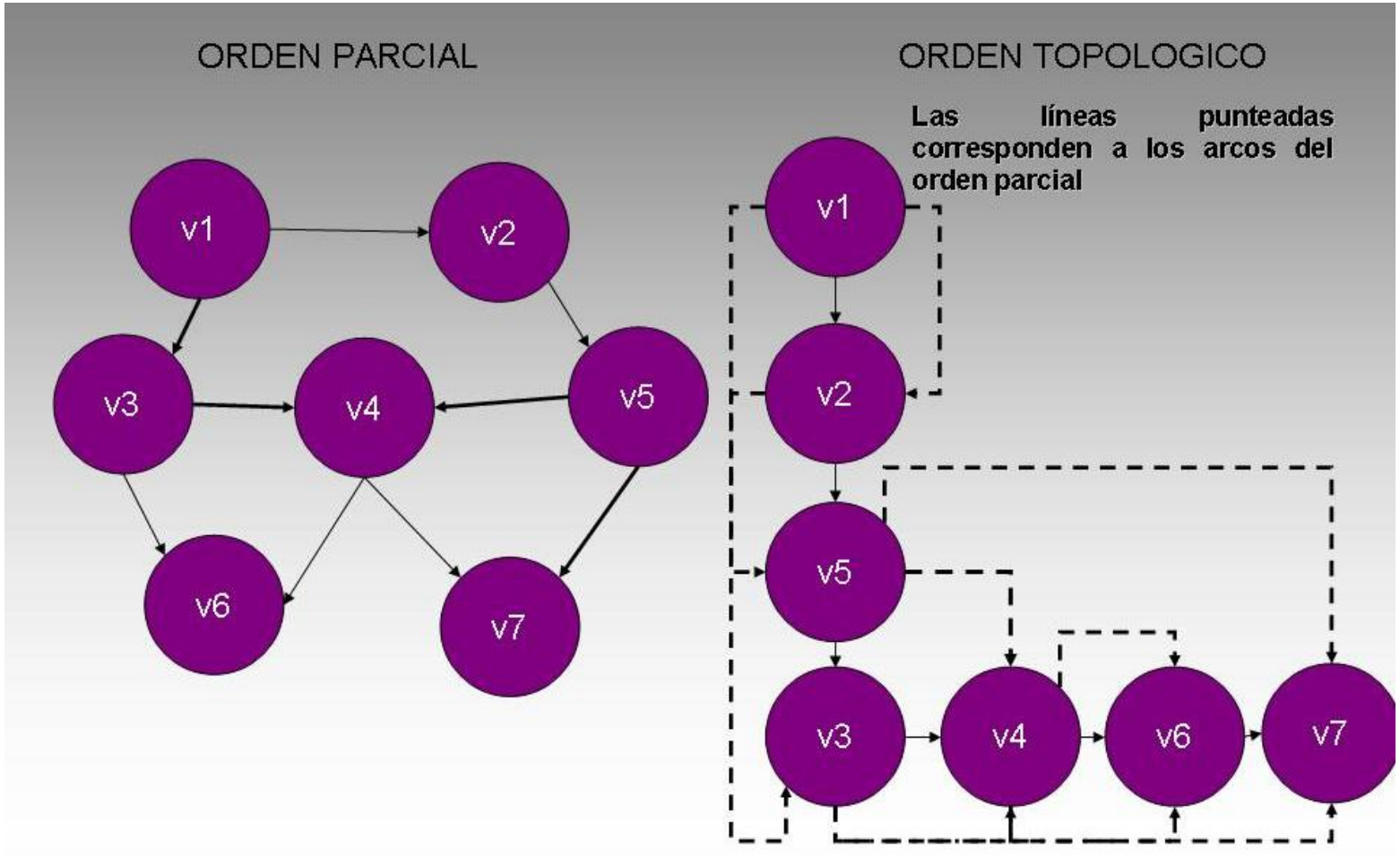
ORDENAMIENTO TOPOLÓGICO



Dos modelos de ordenación topolígica.



ORDENAMIENTO TOPOLÓGICO



GRAFO

Lea el párrafo que aparece abajo y complete las palabras que faltan.

Un grafo [dirigido] es aquel en el que todas sus aristas tienen sentido o dirección.

Un grafo no dirigido es aquel en el que todas sus aristas son [bidireccionales] .

Un grafo donde cada arista tiene asociado un valor se denomina [ponderado]

Un grafo no dirigido es [conexo] si hay un camino entre cada par de vértices.

Un grafo dirigido se denomina [fuertemente] [conexo] si existe un camino desde cualquier vértice a cualquier otro vértice.

EJEMPLO (DFS)-(BFS)

Se puede usar el algoritmo BFS

- Mapas de Google para encontrar rutas óptimas entre puntos turísticos de interés
- Búsquedas de textos en una página web por medio de un Web Crawler;

El algoritmo DFS

- Se puede resolver o simular juegos como laberintos o ajedrez.

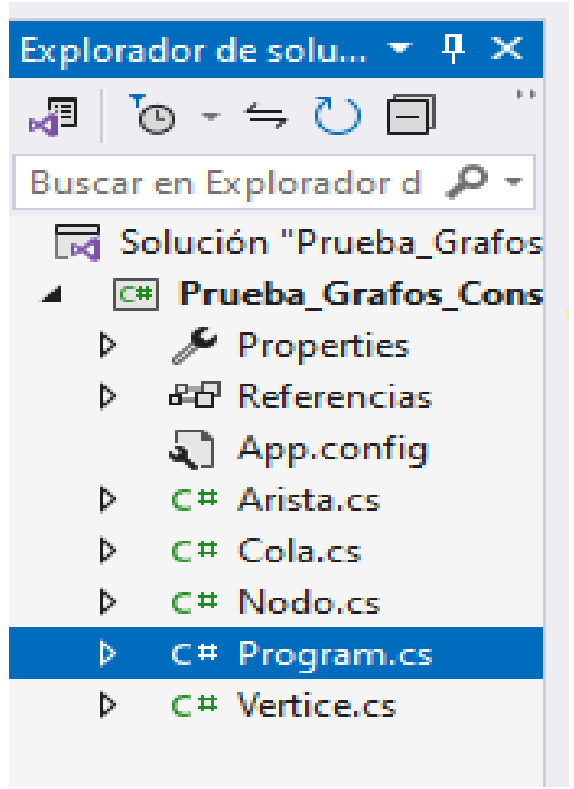
PROBLEMAS

- Resolver la actividad planteada en la plataforma virtual de aprendizajes

PROBLEMAS 1

GRAFO - Consola

Escribir un programa en C# que permita implementar la creación de un grafo. Sin utilizar LIST<>. También se deberá poder eliminar una **arista**, eliminar un **vértice**, recorrer el grafo en **anchura y en profundidad**.
Por último, se deberá mostrar el grafo en pantalla.
-Utilizar las clases: Nodo, Cola, Artista y Vertice



PROBLEMAS 1

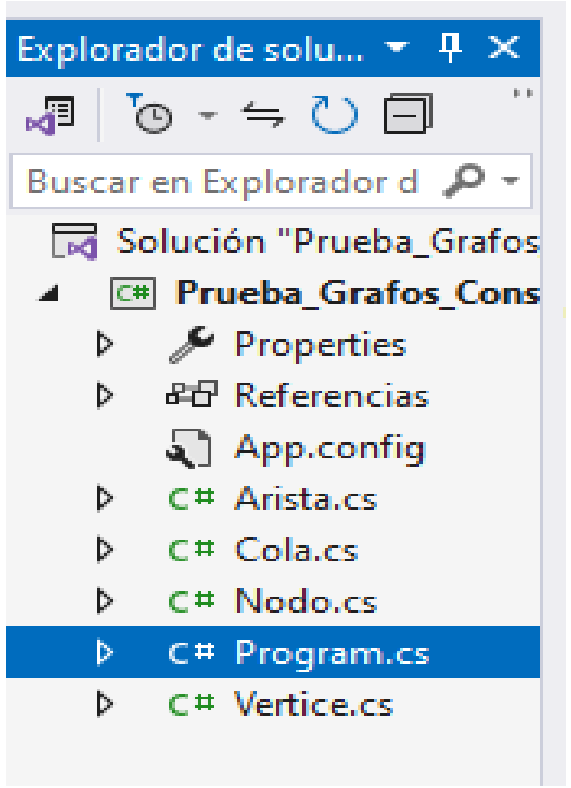
GRAFO - Consola

```
Console.Write("\n|-----------------------------------------------------------|");
Console.Write("\n|                          GRAFOS                           |");
Console.Write("\n|--------------------------|--------------------------------|");
Console.Write("\n| 1. Insertar Vértices     | 5. Recorrer en Anchura         |");
Console.Write("\n| 2. Insertar Aristas      | 6. Recorrer en Profundidad     |");
Console.Write("\n| 3. Eliminar Arista       | 7. Mostrar Grafo               |");
Console.Write("\n| 4. Eliminar Vértice      | 8. Salir                       |");
Console.Write("\n|--------------------------|--------------------------------|");
Console.Write("\nEscoja una Opcion: ");
```

PROBLEMAS 1

GRAFO - Consola

```
|-------------------------------------------------|
|                     GRAFOS                      |
|-----------------------|-------------------------|
| 1. Insertar Vértices  | 5. Recorrer en Anchura  |
| 2. Insertar Aristas   | 6. Recorrer en Profundidad |
| 3. Eliminar Arista    | 7. Mostrar Grafo        |
| 4. Eliminar Vértice   | 8. Salir                |
|-----------------------|-------------------------|
Escoja una Opcion:
```



PROBLEMAS 2

GRAFO - Consola

Ejercicio de c# en consola utilizando métodos. Algoritmos para grafos:

Definiciones, grafos y grafos dirigidos, aplicaciones, representación, matriz de adyacencia, lista de adyacencia, matriz de costos. NO UTILIZAR List<int>. Recorrido: en amplitud (BFS), en profundidad (DFS), ordenamiento topológico y conectividad.

Crear un aplicativo que muestre el siguiente menú en consola:

Utilizando clase Cgrafo, CLista, CVertice

PROBLEMAS 2

GRAFO - Consola

Crear un aplicativo que muestre el siguiente menú en consola:

Utilizando clase Cgrafo, CLista, CVertice

```
Console.WriteLine("**MENU**");
Console.WriteLine("1. Crear grafo");
Console.WriteLine("2. Agregar vertice");
Console.WriteLine("3. Agregar arco");
Console.WriteLine("4. Mostrar vertices");
Console.WriteLine("5. Mostrar grafo (Matriz de Adyacencia)");
Console.WriteLine("6. amplitud (BFS)");
Console.WriteLine("7. profundidad (DFS)");
Console.WriteLine("8. ordenamiento topológico");
Console.WriteLine("9. grafos dirigidos y no dirigidos");
Console.WriteLine("10. lista de adyacencia");
Console.WriteLine("11. matriz de costos");
Console.WriteLine("0. Salir");
Console.Write("opcion: ");
```

PROBLEMAS 3

GRAFO - Form

Este proyecto consiste en la creación de una aplicación **Windows Forms en C#,** diseñada para construir, visualizar y recorrer grafos dirigidos y ponderados. El enfoque está en evitar el uso de estructuras dinámicas como **List<>,** privilegiando el uso de arreglos estáticos para mantener un control más detallado sobre la memoria y el diseño algorítmico.

🔧 **Componentes clave del sistema**

- **Estructura de datos personalizada**:
  - Vertice: almacena información del nodo y un conjunto fijo de aristas.
  - Arista: define una conexión desde un vértice origen hacia uno destino, incluyendo el peso.
  - Grafo: encapsula la lógica para agregar vértices, establecer conexiones y realizar búsquedas.

- **Interfaz principal** (Form1):
  - Permite agregar nodos y definir aristas con pesos.
  - Muestra tablas con la estructura de conexiones y listas de adyacencia.
  - Integra botones para ejecutar los algoritmos BFS y DFS.

PROBLEMAS 3

GRAFO - Form

Recorridos:
•**BFS (Breadth-First Search)**: implementado con una cola basada en arreglo.
•**DFS (Depth-First Search)**: implementado con una pila basada en arreglo.
•Ambos recorridos actualizan visualmente el orden de visita de los nodos.

Visualización gráfica:
•Un segundo formulario (FormVisualizacion) muestra el grafo en un Panel, dibujando nodos como círculos conectados.
•Durante los recorridos, los nodos cambian de color al ser visitados, ofreciendo una experiencia visual del algoritmo en ejecución.

PROBLEMAS 3

GRAFO - Form

Categoría	Elemento / Herramienta	Descripción
Controles de entrada	TextBox (txtNodo, txtDestino, txtPeso, etc.)	Permiten al usuario ingresar nombres de nodos, destino y pesos de aristas.
Botones	Button (btnAgregarNodo, btnAgregarArista, btnBFS, btnDFS, btnVisualizar)	Disparan las acciones de construcción y recorrido del grafo.
Visualización tabular	DataGridView (dgvLista, dgvListaAdyacente, dgvRecorrido)	Muestran los datos del grafo y el orden de recorrido de los nodos.
Visualización gráfica	Panel (panelGrafo, panelVisual)	Espacio para dibujar nodos, aristas y mostrar recorridos animados.
Formularios múltiples	Form1, FormVisualizacion	Separan la lógica de creación y visualización del grafo para modularidad.
Clases personalizadas	Grafo, Vertice, Arista, NodoVisual	Estructuran la lógica y el posicionamiento del grafo sin usar List<>.
Algoritmos implementados	BFS y DFS	Recorridos clásicos adaptados a estructuras estáticas.
Colores y gráficos	Graphics, Brush, Pen	Permiten pintar nodos visitados, aristas, y etiquetas con estilo.

PROBLEMAS 3

GRAFO - Form

Tipo	Nombre de Control	Función específica en el proyecto
TextBox	txtNodo	Ingresar el nombre del nodo (vértice) a agregar.
	txtDestino, txtNodoIZQ, txtNodoDER	Ingresar el nombre del nodo de destino para crear una arista desde el nodo origen.
	txtPeso, txtPesoIZQ, txtPesoDER	Ingresar el valor del peso de la arista o conexión entre nodos.
Button	btnAgregarNodo	Agrega un nuevo vértice al grafo con el nombre especificado.
	btnAgregarArista	Crea una conexión (arista) entre dos nodos con peso definido.
	btnAgregarIZQ, btnAgregarDER	Agrega conexiones manualmente en estructuras tipo árbol (como hijos izquierdos o derechos).
	btnListaAdyacente	Muestra en tabla las conexiones directas (adyacentes) del nodo seleccionado.
	btnBFS, btnDFS	Ejecutan los algoritmos de recorrido BFS y DFS respectivamente, mostrando el orden de visita.
	btnVisualBFS, btnVisualDFS	Visualizan animadamente los recorridos BFS o DFS en el panel gráfico.
	btnVisualizar	Abre un formulario adicional (FormVisualizacion) para mostrar el grafo gráficamente.
DataGridView	dgvLista	Lista principal de los nodos y sus hijos izquierdo/derecho con sus respectivos pesos.
	dgvListaAdyacente	Muestra las conexiones adyacentes (nodos conectados directamente) desde un nodo dado.
	dgvRecorrido	Visualiza el orden de visita de los nodos durante BFS o DFS (modo texto).
Panel	panelGrafo	Dibuja dinámicamente los nodos y aristas del grafo usando objetos Graphics.
	panelVisual	Panel ubicado en el segundo formulario (FormVisualizacion) con función exclusiva de dibujo limpio.
Timer *(opcional)*	—	Puede usarse para animar el recorrido paso a paso si se desea control de tiempo entre visitas.
Clases Personalizadas	Vertice, Arista	Representan nodos y conexiones del grafo, evitando uso de listas.
	Grafo	Encapsula la lógica del conjunto de vértices y métodos BFS/DFS.
	NodoVisual	Guarda el nombre y la posición gráfica de cada nodo para vincular lógica y visualización.
Form	Form1	Formulario principal de entrada de datos, creación del grafo y botones de control.
	FormVisualizacion	Formulario secundario que renderiza el grafo completo en una vista separada.

PROBLEMAS 3

GRAFO - Form

Form1

Nodo A

IZQ A

DER D

Peso

6

4

Agregar IZQ

Agregar DER

AGREGAR NODO

	NODO	NODO_IZQ	PESO_IZQ	NODO_DER	PESO_DER
▶	A	A	2	B	4
	B	B	1	D	3
	C	A	6	D	4
	D				
	C				

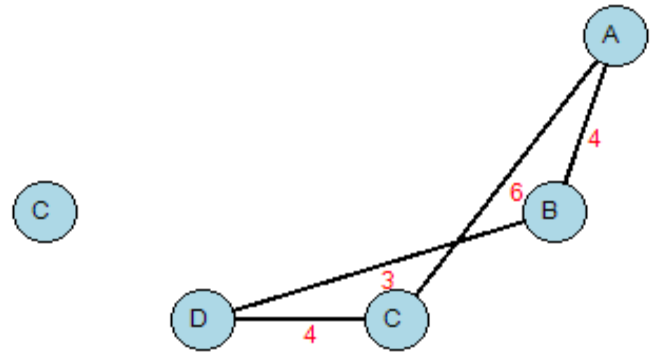
Lista Adyacente

BFS

DFS

	NODO	PESO
▶	A	2
	B	4

	NODO
▶	A
	B
	D

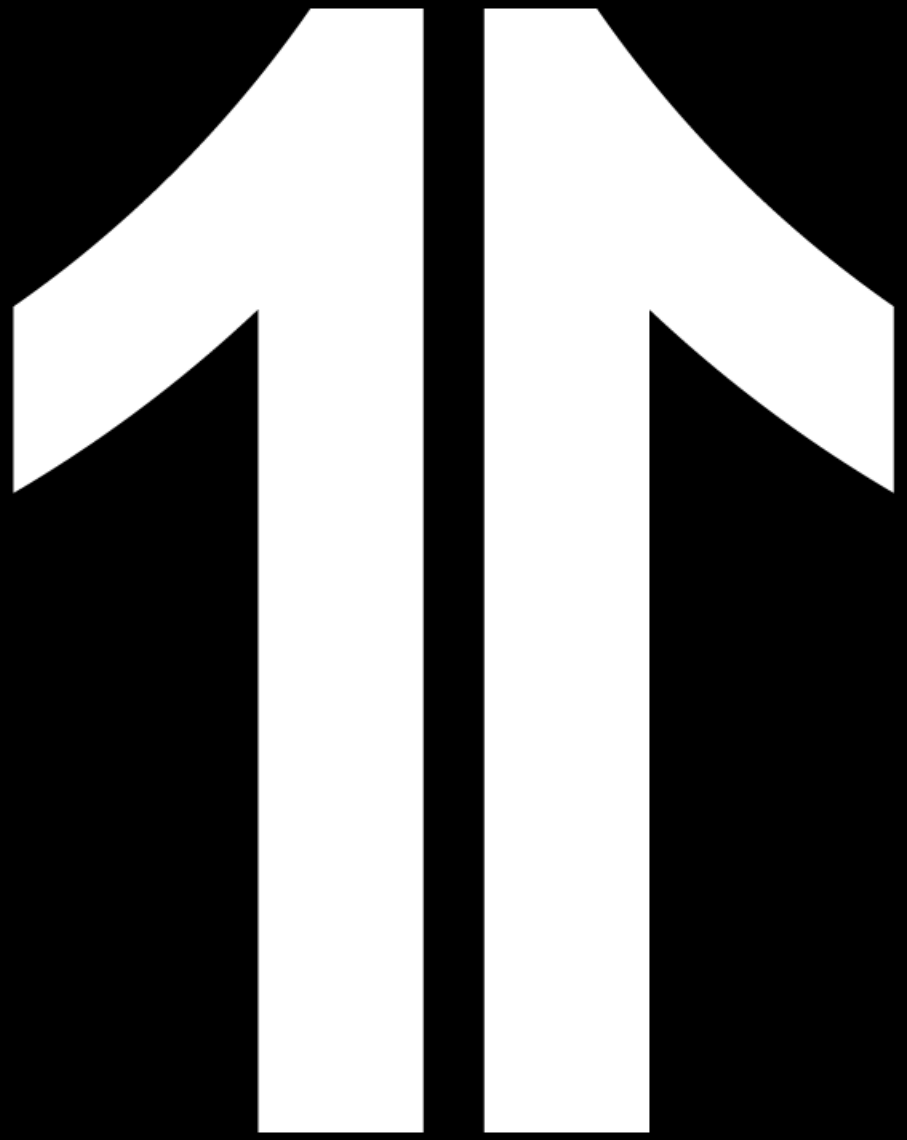


¿Preguntas o comentarios?

¿Preguntas o comentarios?

BIBLIOGRAFIA REFERENCIAL

- Ceballos Sierra, F. Microsoft C#: Curso de Programación (2a.ed.) 2014 https://elibronet.eu1.proxy.openathens.net/es/lc/upnorte/titulos/106417
- Cesar Liza Avila; Estructura de datos con C/C++



GRACIAS


UPN
UNIVERSIDAD
PRIVADA
DEL NORTE