# Manual del Laboratorio 3A

## 2024

# ÍNDICE

# Manual del Laboratorio 3A

(Elaborado por Johan Baldeón)

## 1 Introducción

Además de los TADs como listas enlazadas, pilas, colas y árboles, se tiene también grafos. Los grafos son estructuras matemáticas que permiten modelar relaciones entre objetos. Este manual explora el TAD de grafos y la forma de cómo implementarlos en C++/CLI utilizando las bibliotecas de .NET.

## 2 Conceptos Básicos

### 2.1 ¿Qué es un Grafo?

Un grafo es una estructura compuesta por un conjunto de nodos (o vértices) y un conjunto de aristas (o enlaces) que conectan pares de nodos. Dependiendo de la naturaleza de las aristas, los grafos pueden clasificarse en:

- **Grafos Dirigidos**: Las aristas tienen una dirección, es decir, van de un nodo a otro en una sola dirección.
- **Grafos No Dirigidos**: Las aristas no tienen dirección, lo que significa que la relación entre los nodos es bidireccional.

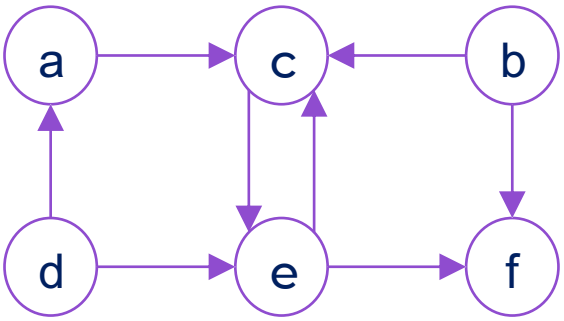


*Figura 1: Ejemplo de grafo dirigido.*

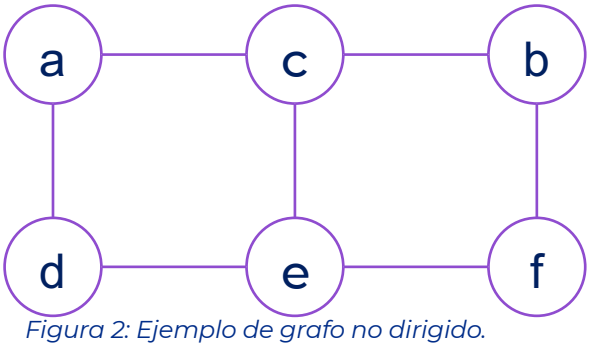


*Figura 2: Ejemplo de grafo no dirigido.*

## 3 Implementación de Grafos en C++/CLI

### 3.1 Representación del Grafo

En C++/CLI, podemos representar un grafo de varias maneras, pero las dos más comunes son mediante una **matriz de adyacencia** y una **lista de adyacencia**.

**Matriz de Adyacencia**: Es una matriz cuadrada n x n donde n es el número de nodos en el grafo. Un valor en la posición (i, j) indica si hay una arista entre el nodo i y el nodo j.
**Lista de Adyacencia**: Es un array de listas donde cada lista contiene los nodos adyacentes a un nodo específico.

## 3.2 Implementación de una Lista de Adyacencia en C++/CLI

A continuación, se presenta un ejemplo de implementación de un grafo dirigido utilizando una lista de adyacencia en C++/CLI.

```
#include "pch.h"

using namespace System;
using namespace System::Collections::Generic;

public ref class Graph
{
    Dictionary<int, List<int>^>^ adjList;

public:
    Graph()
    {
        adjList = gcnew Dictionary<int, List<int>^>();
    }

    // Método para añadir una arista al grafo
    void AddEdge(int src, int dest)
    {
        if (!adjList->ContainsKey(src))
            adjList[src] = gcnew List<int>();
        adjList[src]->Add(dest);
    }

    // Método para mostrar el grafo
    void DisplayGraph()
    {
        Console::WriteLine("Lista de Adyacencia del Grafo:");
        for each (KeyValuePair<int, List<int>^> kvp in adjList)
        {
            Console::Write("{0} -> ", kvp.Key);
            for each (int val in kvp.Value)
            {
                Console::Write("{0} ", val);
            }
            Console::WriteLine();
        }
    }
};

int main(array<System::String^>^ args)
{
    Graph^ graph = gcnew Graph();
    graph->AddEdge(0, 1);
    graph->AddEdge(0, 4);
    graph->AddEdge(1, 2);
    graph->AddEdge(1, 3);
    graph->AddEdge(1, 4);
    graph->AddEdge(2, 3);
```

```
    graph->AddEdge(3, 4);

    graph->DisplayGraph();
    return 0;
}
```