Taller 4 Grafos

Santiago Hernandez Barbosa

Materia: Estructuras de datos
Profesor: John Corredor Franco

20/05/2025
Bogotá, D. C.

Tabla de contenido

1. Introducción

Los grafos son una herramienta muy útil para representar relaciones entre diferentes elementos. A diferencia de otras estructuras de datos más simples, como listas o árboles, los grafos permiten modelar conexiones más complejas, como las que se dan en redes sociales, rutas de transporte o vínculos entre artículos científicos.

Este taller está dividido en dos partes. En la primera se analizan las propiedades de un grafo a partir de su matriz de adyacencia. A través de preguntas puntuales, se estudian conceptos importantes como la existencia de ciclos, los vértices fuente y sumidero, los componentes fuertemente conectados y otras propiedades que permiten entender mejor la estructura del grafo.

En la segunda parte se desarrolla un sistema que simula las citaciones entre artículos académicos. Cada artículo se representa como un nodo, y las referencias entre ellos como conexiones dirigidas. El objetivo es implementar funciones que permitan identificar el artículo más citado, calcular un índice de referenciación, encontrar citaciones indirectas y analizar cómo se conectan los grupos de artículos al eliminar uno del sistema. Para esto, se diseña una solución basada en TADs (Tipos Abstractos de Datos), se definen las relaciones entre ellos y se programan los algoritmos en lenguaje C++.

Con este taller se busca aplicar los conocimientos sobre grafos a un caso práctico, reforzando tanto la parte teórica como la capacidad para resolver problemas usando estructuras de datos más avanzadas.

2. Parte I: Análisis teórico del grafo

1. ¿Es un grafo cíclico o acíclico? en caso de ser cíclico, describa todos los ciclos en el grafo.

- Las cuatro matrices son grafos cíclcos

- **G_1** Ciclos:
  $2 \rightarrow 3 \rightarrow 5 \rightarrow 2$
- **G_2** Ciclos:
  $2 \rightarrow 4 \rightarrow 5 \rightarrow 2$
  $2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 2$
- **G_3** Ciclos
  $2 \rightarrow 4 \rightarrow 3 \rightarrow 2$
  $3 \rightarrow 5 \rightarrow 4 \rightarrow 3$
- **G_4** Ciclos
  $3 \rightarrow 4 \rightarrow 2 \rightarrow 3$
  $1 \rightarrow 5 \rightarrow 3 \rightarrow 1$
  $1 \rightarrow 5 \rightarrow 4 \rightarrow 2 \rightarrow 3 \rightarrow 1$

2. ¿Hay vértices fuente? ¿Cuáles son?

- (Un vértice fuente es el que no tiene aristas entrantes.)

- **G_1**: { 1 }
- **G_2**: { 1 }
- **G_3**: { 1 }
- **G_4**: ∅

3. ¿Hay vértices sumidero? ¿Cuáles son?

- (Un vértice sumidero es el que no tiene aristas salientes.)

- **G_1**: { 4 }
- **G_2**: ∅
- **G_3**: ∅
- **G_4**: ∅

4. ¿Cuáles son los vértices descendientes de 2?

- (Todos los vértices alcanzables desde el vértice 2, excluyendo el mismo 2.)

- **G_1**: { 3, 4, 5 }
- **G_2**: { 3, 4, 5 }
- **G_3**: { 3, 4, 5 }
- **G_4**: { 1, 3, 4, 5 }

5. ¿Cuántos componentes fuertemente conectados hay en el grafo?

- (Número de conjuntos de vértices donde cada par es mutuamente alcanzable.)

- **G_1**: 3 componentes
  - { 1 }, { 2, 3, 5 }, { 4 }
- **G_2**: 2 componentes
  - { 1 }, { 2, 3, 4, 5 }
- **G_3**: 2 componentes
  - { 1 }, { 2, 3, 4, 5 }
- **G_4**: 1 componente
  - { 1, 2, 3, 4, 5 }

Pregunta	**Respuesta**	**Justificación**
Si un grafo no dirigido y conectado contiene un camino de Hamilton, éste es exactamente igual a su correspondiente camino de Euler.	F	Un camino de Hamilton recorre vértices, un de Euler recorre aristas; son conceptos distintos.
Un grafo dirigido de N vértices, con un vértice fuente y un vértice sumidero, puede estar fuertemente conectado.	F	Si hay un vértice fuente (sin entradas) o sumidero (sin salidas) no todos los pares son mutuamente alcanzables.
Sólo se puede definir camino(s) o circuito(s) de Euler en un grafo con un único componente conectado.	V	Un circuito o camino de Euler cubre todas las aristas en un solo recorrido, por lo que el grafo debe estar conectado.
La matriz de adyacencia de un grafo no dirigido es simétrica por la diagonal.	V	En un grafo no dirigido, cada arista i–j aparece también como j–i, lo que hace la matriz simétrica.
Un grafo dirigido está fuertemente conectado cuando existe un camino entre cada par de vértices, sin tener en cuenta las direcciones de las conexiones.	F	Eso describe conectividad débil; la fuerte requiere respetar las direcciones al ir y volver entre cada par.
El algoritmo de Dijkstra genera un árbol de recubrimiento de costos mínimos, así como el algoritmo de Prim.	F	Dijkstra halla caminos mínimos desde una fuente, Prim construye un árbol de expansión mínimo.
La matriz de caminos de un grafo con N vértices y M aristas se calcula sumando la matriz identidad de tamaño NxN con la matriz de adyacencia.	F	La matriz de caminos (cierre transitivo) requiere más que una suma simple; la suma I+A sólo añade caminos de longitud ≤ 1.

Si la matriz de adyacencia de un grafo es una matriz diagonal inferior, se puede decir que el grafo es dirigido.	V	Una matriz estrictamente triangular inferior no es simétrica, por lo que no puede corresponder a un grafo no dirigido.

3. Parte II: Sistema de Citaciones Académicas (Grafo dirigido)

3.1 Descripción del problema

En la fase inicial de un proyecto de investigación, uno de los pasos fundamentales es construir el estado del arte, es decir, recopilar y analizar los trabajos previos que se han publicado en la materia. Cada artículo suele citar a otros para fundamentar sus aportes y mostrar la continuidad del conocimiento. Llevar un control manual de todas estas referencias rápidamente se vuelve complejo a medida que crece el número de publicaciones.

Para facilitar este análisis, podemos representar las citaciones como un grafo dirigido. En este modelo, cada nodo equivale a un artículo y cada arco dirigido indica que el artículo de origen cita al de destino. Por ejemplo, si el artículo A cita a B y a C, en el grafo aparecerán aristas A→B y A→C. El resultado es un "mapa de citaciones" que permite visualizar a simple vista qué trabajos reciben más referencias y cómo se vinculan entre sí.

El objetivo de este sistema es ofrecer herramientas para explorar ese grafo de citaciones de manera automática. En particular, se busca:

1. **Identificar el artículo más citado**,
2. **Contar los grupos de artículos relacionados** al eliminar una obra de interés,
3. **Calcular un índice de referenciación** para medir la relevancia de cada artículo, y
4. **Contar las citaciones indirectas** que un artículo genera a través de sus referencias.

Con ello, los investigadores podrán entender mejor la estructura de sus redes de citación y enfocar sus esfuerzos en los trabajos de mayor impacto.

4. Diseño del sistema

4.1 Diseño textual de TADs

- TAD: Artículo

Datos mínimos:

- id: número entero
  Identificador único del artículo.

- referencias: lista de números enteros
  Contiene los id de los artículos que este artículo cita.

Operaciones:

- obtenerId(): devuelve el id del artículo.
- obtenerReferencias(): devuelve la lista de ids citados.
- agregarReferencia(idDest): añade idDest a la lista de referencias.

-TAD: MapaCitaciones

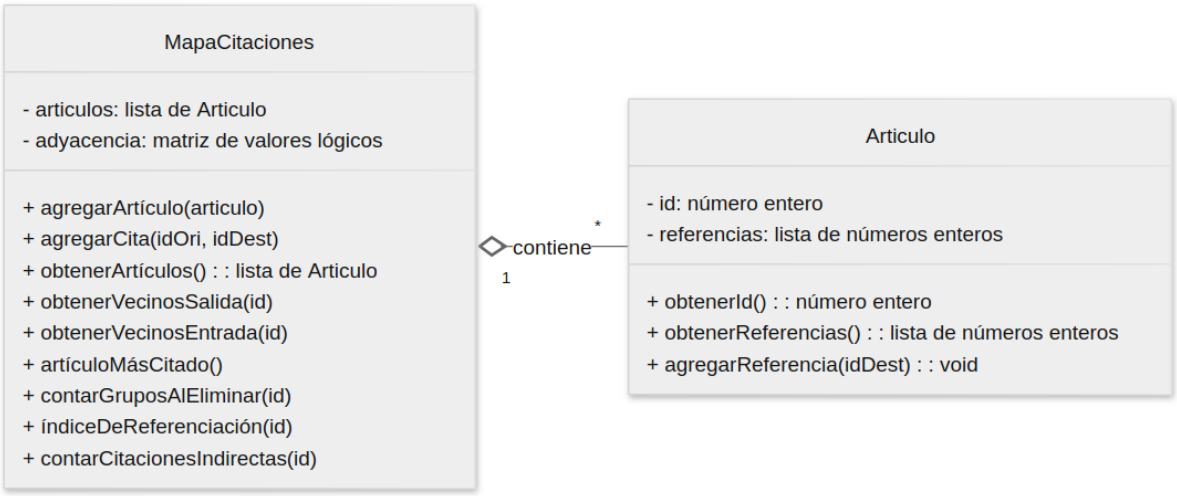
Datos mínimos:

- articulos: lista de Artículo, Conjunto de todos los artículos en el sistema.
- adyacencia: matriz de valores lógicos, Matriz cuadrada en la que adyacencia[i][j] es verdadero si el artículo con id i cita al artículo con id j.

Operaciones:

- agregarArtículo(art): añade un Artículo al conjunto.
- agregarCita(idOri, idDest): marca adyacencia[idOri][idDest] = verdadero.
- obtenerArtículos(): devuelve la lista completa de Artículo.
- obtenerVecinosSalida(id): devuelve lista de ids j tales que adyacencia[id][j] es verdadero.
- obtenerVecinosEntrada(id): devuelve lista de ids i tales que adyacencia[i][id] es verdadero.
- artículoMásCitado(): devuelve el id del artículo con más entradas en su columna de adyacencia.
- contarGruposAlEliminar(id): elimina temporalmente el vértice id (y sus aristas) y devuelve el número de componentes conexos resultantes.
- índiceDeReferenciación(id): calcula (nEntrantes(id)) / (½·nSalientes(id)) y devuelve el valor numérico.
- contarCitacionesIndirectas(id): devuelve la cantidad de vértices alcanzables en dos saltos o más, sin contar las referencias directas ni id.

5. Diagrama de relación entre TADs



6. Implementación de algoritmos (en C++)

6.1 Algoritmo 1 – Artículo más citado

Este método identifica el artículo que recibe más citas (aristas entrantes) en el grafo.

Código en C++:

```cpp
// 1) devuelve id del artículo con más citas entrantes
int MapaCitaciones::articuloMasCitado() const {
    int n = adyacencia.size();
    std::vector<int> cuenta(n, 0);
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            if (adyacencia[i][j]) ++cuenta[j];

    int maxC = -1, mejorId = -1;
    for (int i = 0; i < n; ++i) {
        if (cuenta[i] > maxC) {
            maxC    = cuenta[i];
            mejorId = articulos[i].obtenerId();
        }
    }
    return mejorId;
}
```

Recorre la matriz para contar cuántas veces cada columna es true y elige el índice con mayor cuenta.

6.2 Algoritmo 2 – Grupos de artículos al eliminar uno

Este método elimina (conceptualmente) un artículo dado y cuenta cuántos grupos de artículos conectados quedan, considerando el grafo como no dirigido.

```
// 2) elimina id y cuenta componentes conexos (grafo no dirigido)
int MapaCitaciones::contarGruposAlEliminar(int id) const {
    int rem = indexOf(id), n = adyacencia.size();
    if (rem < 0) return 0;
// eliminar id
    std::vector<bool> vis(n, false);
    int grupos = 0;
    for (int i = 0; i < n; ++i) {
        if (i == rem || vis[i]) continue;
        ++grupos;
        std::queue<int> q;
        q.push(i); vis[i] = true;
        while (!q.empty()) {
            int u = q.front(); q.pop();
            for (int v = 0; v < n; ++v) {
                if (v == rem || vis[v]) continue;
                if (adyacencia[u][v] || adyacencia[v][u]) {
                    vis[v] = true;
                    q.push(v);
                }
            }
        }
    }// contar componentes conexos
    // (grafo no dirigido)
    return grupos;
}
```

Trata el grafo como no dirigido (revisa ambas direcciones) y cuenta componentes conexos con BFS, omitiendo el vértice eliminado.

6.3 Algoritmo 3 – Índice de referenciación

Este método calcula, para un artículo dado, el índice de referenciación definido como *número de artículos que citan al artículo / (0.5* número de artículos citados por el artículo)*

```cpp
// 3) devuelve entrantes / (0.5 * salientes)
double MapaCitaciones::indiceDeReferenciacion(int id) const {
    int idx = indexOf(id), n = adyacencia.size();
    if (idx < 0) return 0.0;

    int ent = 0, sal = 0;
    for (int j = 0; j < n; ++j) {
        if (adyacencia[j][idx]) ++ent;
        if (adyacencia[idx][j]) ++sal;
    } // contar entrantes y salientes
    if (sal == 0) return 0.0;
    return double(ent) / (0.5 * sal);
}
```

Divide el número de artículos que citan a id entre la mitad de los que id cita.

6.4 Algoritmo 4 – Citaciones indirectas

Este método cuenta cuántos artículos se alcanzan desde el artículo dado en dos o más saltos (excluye las citas directas).

```cpp
// 4) cuenta nodos alcanzables en ≥2 saltos desde id
int MapaCitaciones::contarCitacionesIndirectas(int id) const {
    int idx = indexOf(id), n = adyacencia.size();
    if (idx < 0) return 0;

    std::vector<int> nivel(n, -1);
    std::queue<int> q;
    nivel[idx] = 0; q.push(idx); // nivel de id es 0

    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (int v = 0; v < n; ++v) {
            if (adyacencia[u][v] && nivel[v] == -1) {
                nivel[v] = nivel[u] + 1;
                q.push(v);
            }
        }
    } // contar nodos alcanzables

    int cuenta = 0;
    for (int i = 0; i < n; ++i)
        if (nivel[i] >= 2) ++cuenta;
    return cuenta;
}
```

Utiliza BFS para calcular la distancia (en número de saltos) desde id y cuenta sólo los nodos a dos o más saltos (excluye las referencias directas).

7. Conclusiones

En este taller se ha fortalecido la comprensión teórica de los grafos al identificar ciclos, vértices fuente y sumidero, así como componentes fuertemente conectados a partir de matrices de adyacencia. Además, se ha aplicado esa teoría al diseño de TADs (Articulo y MapaCitaciones) y a la implementación en C++ de algoritmos que determinan el artículo más citado, cuentan grupos tras eliminar un nodo, calculan un índice de referenciación y detectan citaciones indirectas.

La separación entre la interfaz (especificación de TADs) y la implementación (archivos .h y .cpp) ha promovido un código modular y legible. El caso práctico de las citaciones académicas ha demostrado la eficacia de los grafos para analizar relaciones complejas en entornos de investigación, facilitando la identificación de trabajos clave y conexiones no evidentes.