Pontificia Universidad Javeriana

Taller 04: Grafos

Juan Pabon Vargas

Estructuras de datos

19 de mayo de 2025

Objetivos

El propósito de este taller es comprender y reforzar los conocimientos previos sobre los grafos y su funcionamiento, tanto en teoría como en la práctica. En la teoría, implica comprender los elementos clave de los grafos y su funcionamiento mediante ejemplos. En la parte práctica, el enfoque es aplicar estos conocimientos mediante la implementación en el contexto de un ejemplo real. Esto también integra conceptos previamente aprendidos, como el diseño con Tipos Abstractos de Datos y la implementación de soluciones basadas en dichos diseños.

Introducción

Los grafos pueden adoptar diversas formas, pero en este taller nos centraremos en grafos dirigidos y no dirigidos, tanto cíclicos como acíclicos. El laboratorio también explora conceptos clave como los componentes fuertemente conexos y los diferentes tipos de vértices, incluyendo fuentes y sumideros.

A diferencia de las estructuras de datos lineales, los grafos son no lineales y reflejan mejor las relaciones complejas del mundo real. En los grafos, los elementos de datos no están organizados en una jerarquía. En cambio, los nodos pueden conectarse a varios nodos simultáneamente. Un grafo es dirigido si sus aristas tienen direcciones específicas y no dirigido si las conexiones fluyen en ambos sentidos. Los grafos cíclicos contienen uno o más bucles que permiten el recorrido desde un nodo hacia sí mismo. Los componentes fuertemente conectados son subconjuntos de un grafo dirigido donde cada nodo es accesible desde todos los

demás nodos dentro del mismo componente. En este contexto, un origen es un nodo con solo aristas salientes, mientras que un sumidero solo tiene aristas entrantes.

El taller comienza con el análisis de cuatro grafos basados en sus matrices de adyacencia, para responder preguntas sobre su estructura y propiedades. Posteriormente, se presenta un conjunto de preguntas de verdadero/falso que abarcan conceptos generales de la teoría de grafos. Finalmente, se pide a los estudiantes que diseñen e implementen operaciones específicas relacionadas con grafos en un escenario de ejemplo dado, que incluye el diseño de tipos abstractos de datos (TDA), un diagrama de relaciones y la implementación de varios algoritmos.

SECCIÓN DE PREGUNTAS Y VERDADERO/FALSO

A continuación se trabajarán las preguntas de análisis sobre grafos dado sus matrices de adyacencia, y luego la sección de falso/verdadero correspondiente.

$$\mathbf{G_1} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{G_2} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{G_3} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad \mathbf{G_4} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Imagen 1, Matrices de ejercicio

Modelado de cada grafo:

A continuación, se presenta el modelado de cada grafo dado por el enunciado según sus matrices de adyacencia.

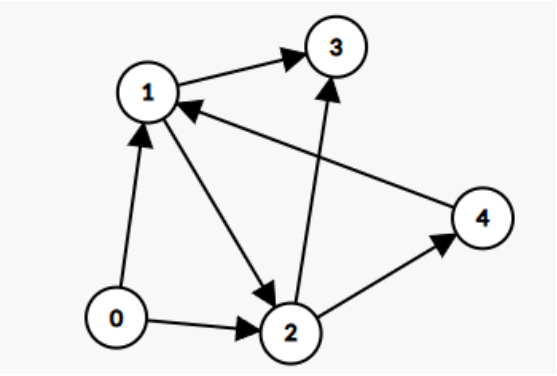

Imagen 2, Grafo modelado G1 (fuente propia)

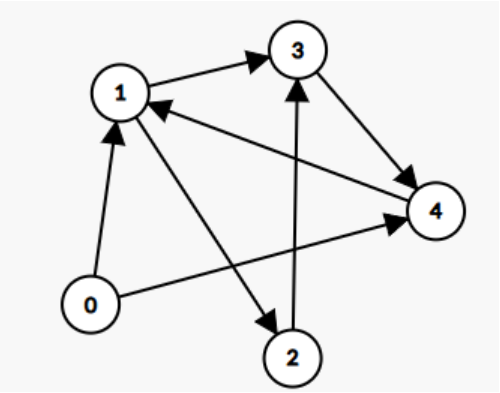

Imagen 3, Grafo modelado G2 (fuente propia)

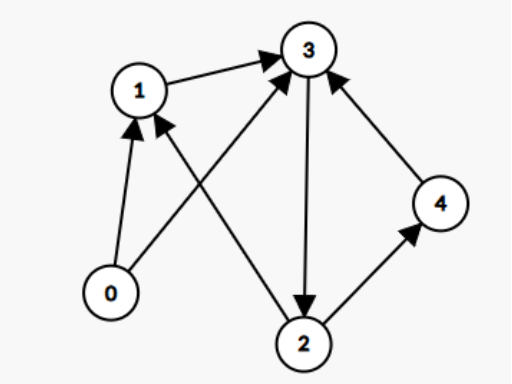

Imagen 4, Grafo modelado G3 (fuente propia)

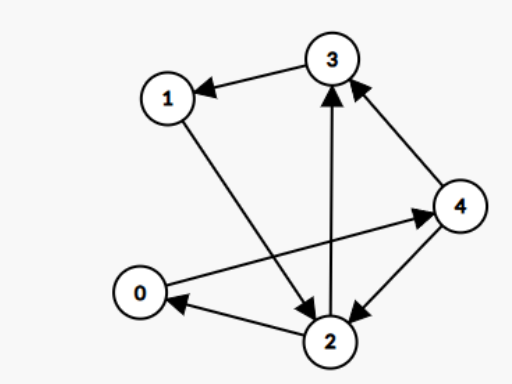

Imagen 5, Grafo modelado G4 (fuente propia)

Preguntas

A continuación se presentarán ahora las preguntas con sus respectivas respuestas, algunas representadas como varias respuestas, una para cada grafo a la que corresponda (cada respuesta está marcada con el título del grafo usado (ej G1, G2 etc))

1.- Responda las siguientes preguntas teniendo en cuenta el grafo representado por la siguiente matriz de adyacencia:

1. ¿Es un grafo cíclico o acíclico? En caso de ser cíclico, describa todos los ciclos en el grafo.
    - G1: Cíclico, con ciclo: {(1,2),(2,4),(4,1)}
    - G2: Cíclico, con ciclo: {(1,3),(3,4),(4,1)}
    - G3: Cíclico, con ciclos: {(4,1),(3,2),(2,4)}, {(2,1),(1,3),(3,2)}
    - G4: Cíclico, con ciclos: {(2,3),(3,1),(1,2)}, {(2,0),(0,4),(4,2)} y {(4,3),(3,1),(1,2),(2,0),(0,4)}
2. ¿Hay vértices fuente? ¿Cuáles son?
    - G1: Si posee un vértice fuente: Vértice 0
    - G2: Si posee un vértice fuente: Vértice 0
    - G3: Si posee un vértice fuente: Vértice 0
    - G4: No posee vértices fuente
3. ¿Hay vértices sumidero? ¿Cuáles son?
    - G1: Si posee un vértice sumidero: Vértice 3
    - G2: No posee vértices sumidero
    - G3: No posee vértices sumidero
    - G4: No posee vértices sumidero
4. ¿Cuáles son los vértices descendientes de 2?
    - G1: 1, 3 y 4
    - G2: 1, 3 y 4
    - G3: 1, 3 y 4
    - G4: 0, 1, 3 y 4
5. ¿Cuántos componentes fuertemente conectados hay en el grafo?
    - G1: 3: (1, 2 y 4)
    - G2: 4: (1, 2, 3 y 4)
    - G3: 4: (1, 2, 3 y 4)
    - G4: 4: (0, 1, 2, 3 y 4)

Falso/Verdadero

6. Si un grafo no dirigido y conectado contiene un camino de Hamilton, éste es exactamente igual a su correspondiente camino de Euler.
    - **Falso**
      Dado que puede haber más aristas de las necesarias para formar un camino Hamilton (incluido un número impar de aristas), esto puede impedir la existencia de un camino de Euler.
7. Un grafo dirigido de N vértices, con un vértice fuente y un vértice sumidero, puede estar fuertemente conectado.
    - **Falso**
      Estos vértices implican que solo pueden alcanzar los otros vértices o que los otros vértices lo alcanzan pero no puede hacer las dos. Eso no puede hacer un grafo fuertemente conectado.

8. Sólo se puede definir camino(s) o circuito(s) de Euler en un grafo con un único componente conectado.
    - **Verdadero**
      Si un grafo tiene varios componentes conectados, no se puede definir camino(s) o circuito(s) de Euler porque se necesitaría saltar al otro componente y esa arista no existe para llegar al componente.
9. La matriz de adyacencia de un grafo no dirigido es simétrica por la diagonal.
    - **Verdadero**
      El camino de A a B también aplica de B a A, haciendo la matriz de adyacencia simétrica por la diagonal.
10. Un grafo dirigido está fuertemente conectado cuando existe un camino entre cada par de vértices, sin tener en cuenta las direcciones de las conexiones.
    - **Falso**
      Cada vértice debe tener una entrada y salida para que un grafo dirigido esté fuertemente conectado.
11. El algoritmo de Dijkstra genera un árbol de recubrimiento de costos mínimos, así como el algoritmo de Prim.
    - **Verdadero**
      Ambos crean un árbol de caminos más cortos desde un nodo hasta haber pasado por todos los otros nodos.
12. 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 del grafo.
    - **Verdadero**
      La matriz de caminos se calcula sumando la matriz identidad a la matriz de adyacencia del grafo.
13. Si la matriz de adyacencia de un grafo es una matriz diagonal inferior, se puede decir que el grafo está dirigido.
    - **Verdadero**
      Si la matriz es diagonal inferior, eso implica que no es simétrica. Una matriz que no es simétrica no es dirigida.

SECCIÓN DISEÑO E IMPLEMENTACIÓN:

Diseño del sistema y el (los) TAD(s) solicitado(s)

A continuación se presentarán los TADS solicitados. Uno de estos TADs es el TAD de Grafo utilizado en clase. Se utilizó la plantilla de especificación de TADs vista en clase para el diseño.

TAD NodoGrafo
Datos mínimos:

- Autores: Autores de la cita
- ID: Identificador del artículo
- RevistaPubli: Revista donde se publicó la cita
- Volumen: Cadena de caracteres que define la edición o entrega de la revista en el que se publicó el artículo
- AnioPublic: Año de publicación
- visitado: booleano que indica si ya se paso por este artículo

Operaciones:

- get/set correspondientes

TAD Grafo
Datos mínimos:

- numNodos: cantidad de nodos en el grafo
- Nodos: Vector de nodos del grafo
- Aristas: Vector de pares que contiene las aristas entre 2 nodos (Teniendo el valor de inicio al comienzo, luego a dónde se dirige la arista)
- MatAdj: Matriz de adyacencia del grafo (vector de vectores booleanos), define conexiones entre todos los nodos

Operaciones:

- get/set correspondientes
- FormarAristas(): Llena la lista de aristas con las relaciones adecuadas por medio de la matriz de adyacencia.
- MayCit(): Retorna la cita con mayor cantidad de menciones.
- grupArt(interes): Retorna la cantidad de artículos que rodean un articulo de interes
- IndReferenciacion(ABuscar): Retorna el valor del índice de referenciación de una cita (flotante)
- CitInd(NodI): Retorna un entero con la cantidad de citas indirectas de un artículo inicial
- dirToNoDir(matAd): transforma una matriz de adyacencia de un grafo direccional a uno no direccional.
- remEdge(matAd, id): borra el nodo dado y sus aristas con los nodos con que está conectado.

Diagrama de relación entre TADs

A continuación se presenta el diagrama de relación resultado del diseño via TAD producido anteriormente.

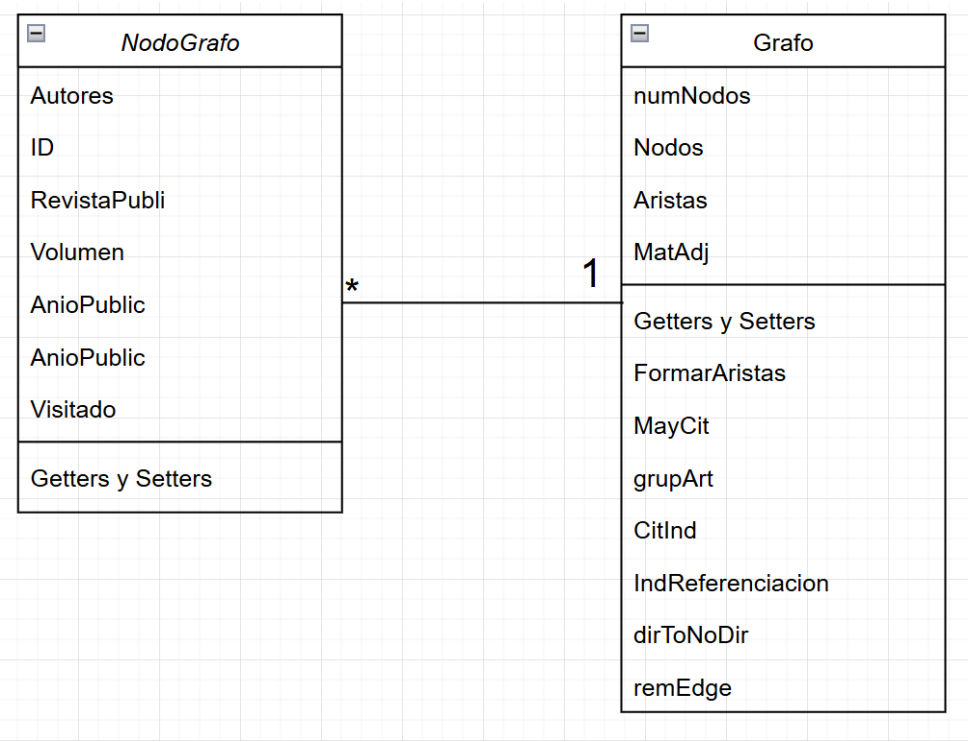

Imagen 6, diagrama de relación resultado de la propuesta de diseño (fuente propia)

Este diagrama muestra una relación uno a muchos entre el grafo y sus respectivos nodos como se propone en el TAD mediante vectores de nodos y aristas.

Algoritmo 1

Pasos a seguir:

1. Viajar por cada nodo, e iniciar un contador para cada nodo de la cantidad de veces en que se repiten
2. Viajar por cada arista
3. Para cada nodo, se revisa si está conectado con otro nodo, con este siendo a dónde se dirige la arista. De ser así, incrementar el contador para dicho nodo
4. Para cada contador de cada nodo, verificar cual contador es mayor, y guardar el índice de dicho contador, que corresponde igualmente al índice del nodo más referenciado
5. El nodo más referenciado es el nodo en el índice de mayor cantidad de repeticiones

---

```
NodoGrafo Grafo::MayCit() {
	vector<int> cantidades;
	NodoGrafo N;
	//Llena el vector de cantidades con la cantidad de repeticiones de cada nodo como nodo llamado
	for(int i=0; i<Nodos.size(); i++) {
		N=Nodos[i];
		for(auto A : Aristas) {
			if(N.getID()==A.second.getID()) {
				cantidades[i]++;
			}
		}
	}
	//Buscará cuál de los nodos tiene el mayor número de repeticiones
	int index=0; //Almacena la posición del nodo en el vector
	int invalue=0; //Almacena el valor en el vector de enteros
	for(int j=0; j<cantidades.size(); j++) {
		if(cantidades[j]>invalue) {
			invalue=cantidades[j];
			index=j;
		}
	}
	//Retorna el nodo en la posición establecida anteriormente
	return Nodos[index];
}
```

Algoritmo 2

Pasos a seguir:

1. Viajar por cada nodo para encontrar el nodo indicado
2. Se elimina el nodo del grafo y sus conexiones con otros nodos
3. se transforma el grafo de un grafo dirigido a un grafo no dirigido por medio de la matriz de adyacencia.
4. viaja por todos los nodos del grafo y toma en cuenta los nodos donde ha pasado para saltarlos.
5. Se toma en cuenta la cantidad de veces que tuvo que viajar por nodos para tener la cantidad de grupos en el grafo que se definen alrededor del articulo de interes.

```
//Toma un grafo guiado y lo transforma en un grafo no guiado
vector<vector<bool>> Grafo::dirToNoDir(vector<vector<bool>> matAd)
{
	vector<vector<bool>> noDirMat = MatAdj;
	//como un grafo no guiado es simétrico, lo que está en m[i][j] tiene que ser igual a lo que está en m[j][i]. Eso se hace aquí.
	for(int i=0;i<numNodos;++i)
	{
```

```
                for(int j=0;j<numNodos;++i)
                {
                        noDirMat[j][i] = noDirMat[i][j];
                }
        }
        return noDirMat;
}
//borra las aristas que contengan el nodo que se va a borrar.
vector<vector<bool>> Grafo::remEdge(vector<vector<bool>> matAd, string id)
{
        int ind = -1;
        //busca el nodo con el id para encontrar su índice
        for(int i=0;i<Nodos.size();++i)
        {
                if(Nodos[i].getId()==id)
                {
                        ind = i;
                }
        }

        //si no encuentra el índice, manda este mensaje
        if(ind==-1)
        {
                cout<<"no se encontró el artículo con ese id"<<endl;
                return;
        }
        //si la encuentra, se asegura que se borren sus conexiones.
        for(int j=0;j<numNodos;++j)
        {
                matAd[i][j]==false;
                matAd[j][i]==false;
        }
}
//encuentra la cantidad de nodos en un componente completo
void Grafo::dfs(int curr, vector<Nodo>& Nodos)
{
        //como empieza con el nodo dado, ya es considerado visitado
        Nodos[curr]->setVisitado(true);
        //visita los otros nodos que hacen parte de este grupo.
        for(int i=0;i<numNodos;++i)
        {
                if(this->MatAdj[curr][i].getVisitado() && !Nodos[i].getVisitado()){
                        dfs(i,Nodos);
                }
        }
}
//encuentra los grupos que se definen alrededor de un artículo dado su id.
void Grafo::grupArt(string id)
```

```
{
	int compCount = 0; //la cantidad de grupos
	vector<vector<bool>> nMat = remEdge(MatAdj,id); //se remueve el artículo dado
	//transforma la matriz de adyacencia a una matriz simétrica de adyacencia.
	vector<vector<bool>> nDirMat = dirToNoDir(nMat, id);

	int i=0; //esto para para no repetir el proceso de DFS por cada nodo y dañar
compCount
	while(i<numNodos)
	{
		//mira si el nodo ya ha sido visitado. Si no, entonces realiza el dfs y se le
suma 1
		a la cuenta de grupos en el grafo.
		if(Nodos[i]->getVisitado()!=1)
		{
			dfs(i,Nodos);
			compCount++;
		}
		i++;
	}
	cout<<"la cantidad de grupos: "<<compCount<<endl;
}
```

Algoritmo 3

Pasos a seguir:

1. Iniciar dos contadores, para cada vez que el artículo es citado y para cada uno de los artículos que el artículo buscado cita
2. Para cada arista, incrementar los contadores correspondientes segun la posicion del artículo en el par (si esta de primero en el par este está citando, de no ser así, está siendo citado)
3. Verificar si las veces que cito es 0
    a. De ser así, solo se entrega la cantidad de veces que se cita el artículo.
    b. De no ser así, se retorna la cantidad de veces que se cita sobre la cantidad de citas que posee el artículo según la fórmula de índice de referenciación (veces que fue citado/veces que cito).

—----------

```
float Grafo::IndReferenciacion(NodoGrafo ABuscar) {
	int secita=0;
	int cito=0;
	//Llena el vector de cantidades con la cantidad de repeticiones de cada nodo como
nodo llamado
	for(auto A: Aristas) {
		if(ABuscar.getID()==A.second.getID()) {
			secita++;
		}
```

```
        }
        for(auto B: Aristas) {
                if(ABuscar.getID()==B.first.getID()) {
                        cito++;
                }
        }
        //Si cito fue 0
        if(cito==0) {
                return secita;
        }
        return secita/cito;
}
```

Algoritmo 4

Pasos a seguir:

1. Aislar, atravesando todas las aristas, las aristas que sean citas directas del nodo elegido
2. Almacenar en un vector estas citas directas, iniciar un contador para la cantidad de citas indirectas al nodo
3. Para cada nodo que se cita por el nodo buscado, buscar sus citas directas, siendo estas las citas indirectas del nodo buscado, de encontrar una cita indirecta aumentar el contador
4. Retornar el contador de citas indirectas

```
int Grafo::CitInd(NodoGrafo NodI) {
        //Inicia con un vector que contendra todas las citas directas
        vector<pair<NodoGrafo,NodoGrafo>>Directas;
        for(auto E : Aristas) {
                if(E.first.getID()==NodI.getID()) {
                        Directas.push_back(E);
                }
        }
        //Tras obtener todas las citas directas, inicia un contador, y cuenta
        //la cantidad de citas conectadas a estas citas
        int citas=0;
        for(auto CitasDirectas : Directas) {
                //Buscará citas directas de estas otras citas indirectas
                for(auto Busca : Aristas) {
                        if(Busca.first.getID() == CitasDirectas.second.getID()) {
                                citas++;
                        }
                }
        }
```

```
    //retorna la cantidad de citas indirectas que tiene el artículo, es decir citas de las citas directas
        return citas;
}
```

Conclusiones

En este laboratorio, se puede comprender mejor los grafos dirigidos. Se adquiere más práctica con ellos y con la identificación de aspectos de este tipo de estructura de datos. La implementación del mapa de citas muestra cómo se utiliza este tipo de estructura de datos en la vida real. Cada algoritmo que se tuvo que desarrollar brindó la oportunidad de aplicar los conocimientos teóricos adquiridos al código para resolver este problema real. Esto también fue una oportunidad para practicar la creación de grafos y los algoritmos que utilizan el grafo y la matriz de adyacencia para resolver problemas.

Resumen

Existen numerosos algoritmos y métodos para resolver problemas con grafos. Para fortalecer la comprensión de estos algoritmos, se exploran ejemplos teóricos y prácticos. Esto incluye el análisis del comportamiento de grafos representados por matrices de adyacencia, la respuesta a preguntas conceptuales de verdadero/falso y la resolución de un caso práctico que se puede ver en la vida real utilizando sistemas basados en grafos.

El taller está estructurado en tres secciones principales: comienza con el análisis de cuatro casos de prueba (cada uno correspondiente a un grafo diferente), sigue con un conjunto de preguntas teóricas de verdadero/falso y concluye con el diseño e implementación de conceptos relacionados con grafos aplicados a un problema real. Por medio de este taller se logra ampliar la comprensión de la lógica y la funcionalidad de los grafos a nivel teórico y práctico.