Título: Taller04 Grafos

Autores: Jeison Camilo Alfonso Moreno, Douglas Díaz

Fecha: 19/05/2025

Curso: Estructuras de Datos

Profesor: John Corredor

---

1. Resumen

¿Qué problema se plantea?
Implementar un conjunto de TADs y algoritmos para el análisis de grafos dirigidos, incluyendo operaciones de conteo de citaciones y componentes conexos, de forma modular y siguiendo una rúbrica establecida.

¿Qué se propone?
Desarrollar un documento de diseño, un plan de pruebas detallado y una implementación modular en C++ de cuatro algoritmos clave sobre grafos dirigidos.

¿Cómo se soluciona?
- Se define el TAD GrafoCitaciones con datos mínimos y operaciones básicas.
- Se implementan los algoritmos: artículo más citado (in-degree máximo), conteo de grupos tras eliminación de un vértice, índice de referenciación y conteo de citaciones indirectas.
- Se organiza el código en módulos (cabecera, implementación, pruebas y Makefile).

Resultados:
Documento de diseño completo, plan de pruebas con casos básicos y avanzados, y código C++ funcional y probado contra el grafo de ejemplo, obteniendo los valores esperados.

2. Introducción

En este taller de estructuras de datos se aborda el manejo de grafos dirigidos mediante el diseño y la implementación de un TAD especializado. Los grafos dirigidos son fundamentales en aplicaciones como análisis de redes, mapas de citaciones académicas y algoritmos de flujo. El objetivo es reforzar conceptos de modularidad, abstracción de datos y pruebas, aplicando técnicas avanzadas de C++ y garantizando la calidad del software.

3. Desarrollo

El desarrollo del taller se estructura en los siguientes componentes:

Módulos de código:
- **GrafoCitaciones.h / GrafoCitaciones.cpp**: definición del TAD con operaciones básicas

(agregar arista, eliminar vértice, obtener adyacencia) y los cuatro algoritmos solicitados.
- **main.cpp**: programa de prueba automático que carga el grafo de ejemplo, ejecuta cada algoritmo y verifica los resultados mostrando PASS/FAIL.
- **Makefile**: reglas para compilar los objetos, enlazar el ejecutable y limpiar artefactos.

Flujo de trabajo:
1. **Diseño**: especificar TAD con datos mínimos y operaciones siguiendo plantilla.
2. **Implementación**: codificar cada módulo en C++ con `using namespace std;` y comentarios claros.
3. **Pruebas**: definir casos para grafos vacíos, unitario, lineal, cíclico y eliminación de vértice.
4. **Documentación**: incluir comentarios en el código y un archivo Word con diseño, pruebas y conclusiones.

Consideraciones de complejidad:
- Las operaciones de recorrido (DFS/BFS) se realizan en O(V+E).
- El cálculo de in-degree y out-degree toma O(V+E) al precomputar adyacencias inversas.
- El conteo de citaciones indirectas es $O(d^2)$ en el peor caso, con d el grado saliente del vértice.

Diseño de TADs y Plan de Implementación

1. Diseño Textual de TADs

TAD GrafoCitaciones
Datos mínimos:
- adj: mapa de artículos a lista de artículos citados (adyacencia saliente).
- invAdj: mapa de artículos a lista de artículos que citan (adyacencia entrante).

Operaciones:
- crearGrafo() → GrafoCitaciones: inicializa ambos mapas vacíos.
- agregarArticulo(art) : void: añade la clave en adj e invAdj si no existe.
- agregarCita(orig, dest) : void: inserta dest en adj[orig] e orig en invAdj[dest].
- eliminarArticulo(art) : void: elimina art de ambos mapas y de todas las listas de adyacencia.
- obtenerArticulos() → Secuencia<Articulo>: devuelve todas las claves de adj.
- obtenerCitasDirectas(art) → Secuencia<Articulo>: devuelve adj[art].
- articuloMasCitado() → Articulo: retorna la clave con mayor tamaño de invAdj.
- contarGruposSinArticulo(art) → Entero: realiza dfs/bfs sobre grafo no dirigido copia sin art.
- indiceReferenciacion(art) → Real: invAdj[art].size() / (0.5 * adj[art].size()).
- contarCitacionesIndirectas(art) → Entero: cuenta nodos a dos saltos no en adj[art] ni art.

2. Descripción de Algoritmos

A continuación se explica de manera breve cada algoritmo según los TAD:
Algoritmo 1: identificarArtMasCitado
- Recorre todos los vértices y utiliza un mapa inverso de adyacencia para contar el número de citas entrantes por artículo.

Algoritmo 2: contarGruposSinArticulo
- Crea una copia del grafo, elimina el artículo dado y luego cuenta componentes conectados en el grafo no dirigido resultante usando DFS o BFS.

Algoritmo 3: indiceReferenciacion
- Para el artículo dado, calcula: #citaciones entrantes / (0.5 * #citaciones salientes).

Algoritmo 4: contarCitacionesIndirectas
- Para cada artículo citado directamente, recorre sus citas y cuenta aquellos que no estén en la lista de citas directas y no sean el artículo original.

4. Plan de Pruebas

Se propone el siguiente plan de pruebas automatizadas para cada operación:
- Prueba unidad: agregarArticulo y agregarCita
  - Caso 1: añadir artículo nuevo; verificar que existe en obtenerArticulos().
  - Caso 2: añadir cita entre artículos; verificar adyacencia y aristas inversas.

- Prueba unidad: articuloMasCitado
  - Caso 1: grafo con un artículo sin citas.
  - Caso 2: grafo con varios artículos; verificar que retorna el de mayor número entradas.

- Prueba unidad: contarGruposSinArticulo
  - Caso: eliminar vértice en grafo con varios componentes; comparar contra conteo manual.

- Prueba unidad: indiceReferenciacion
  - Caso: artículo sin salientes (debe retornar 0).
  - Caso: artículo con varias entradas y salidas; validar fórmula.

- Prueba unidad: contarCitacionesIndirectas
  - Caso: grafo con rutas de longitud 2; verificar exclusión de directas y recuento único.

5. Resultados de Pruebas y Conclusiones

Tras ejecutar el plan de pruebas, todos los casos pasaron satisfactoriamente. A continuación se exponen conclusiones detalladas:

- Robustez de la implementación: gracias a las pruebas unitarias exhaustivas, se asegura la fiabilidad de cada operación ante diferentes escenarios extremos.

- Escalabilidad y complejidad: el uso de mapas de adyacencia e inversa permite operaciones de recorrido en tiempo O(V+E). En grafos dispersos, esto garantiza eficiencia.

- Mantenibilidad y modularidad: la separación entre definición (header) y implementación (cpp) facilita la extensión a nuevos algoritmos sin afectar el núcleo.

- Reflexión sobre mejoras futuras: se podría incorporar manejo de pesos en aristas para algoritmos de caminos mínimos, y paralelizar DFS para grandes volúmenes de datos.

- Aprendizajes: la práctica reforzó la importancia de diseñar TADs claros antes de codificar, y de documentar mediante comentarios y un plan de pruebas formales.