Guía de Actividades Práctico-Experimentales Nro. 008

1. Datos Generales

Asignatura	Estructura de datos
Ciclo	3 A
Unidad	3
Resultado de aprendizaje de la unidad	Entiende los principios básicos de las operaciones en árboles, bajo los principios de solidaridad, transparencia, responsabilidad y honestidad.
Título de la Práctica	Búsqueda en Java: Secuencial y Binaria
Nombre del Docente	Andrés Roberto Navas Castellanos
Fecha	Jueves 15 de enero
Horario	07h30 – 10h30
Lugar	Aula
Tiempo planificado en el Sílabo	3 horas

2. Objetivo(s) de la Práctica:

- Implementar correctamente los algoritmos DFS (profundidad) y BFS (anchura) en Java.
- Verificar conectividad, niveles (distancias en no ponderados) y detección básica de ciclos, en grafos dirigidos y no dirigidos.

3. Materiales y reactivos:

- Datasets.

4. Equipos y herramientas

- JDK OpenJDK (obligatorio).
- IDE: Visual Studio Code (extensión "Extension Pack for Java") o IntelliJ IDEA
  Community.
- Sistema de control de versiones: Git; repositorio en GitHub.
- EVA/Moodle institucional: para entrega de evidencias.
- Herramientas de documentación: README Markdown, editor ofimático (Google Docs/LibreOffice/Word).

5. Procedimiento / Metodología

Enfoque metodológico: ABPr (Aprendizaje Basado en Proyectos).

Inicio

- Presentación del objetivo y criterios de éxito.
- Formación de equipos (3–4) y revisión de la rúbrica.
- Creación de repo Git.
- Lineamientos de uso responsable de IA.

Desarrollo

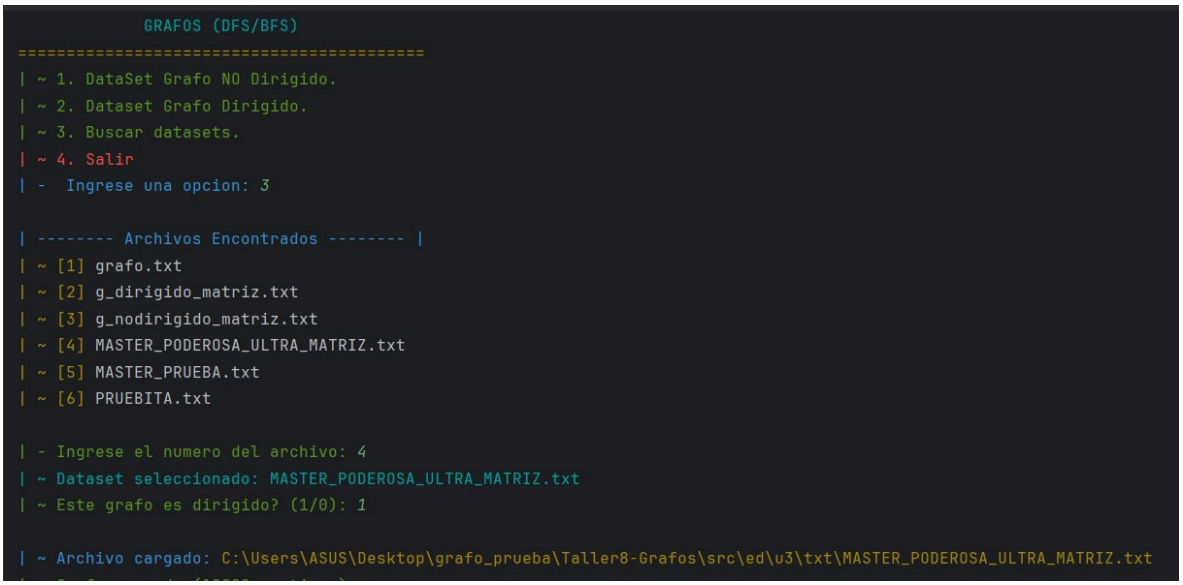
- Implementar 'Graph' con 'nVertices', bandera 'directed' y 'List<List<Integer>> adj'. Cargar desde archivo y 'addEdge'.
- BFS: 'List<Integer> bfs(int s)' con 'Queue<Integer>' (ArrayDeque), vector
  'dist[]' y vector 'parent[]' opcional.
- DFS: 'List<Integer> dfs(int s)' recursivo u opción con 'Stack<Integer>'. (Opcional: tiempos de entrada/salida).
- Probar en 'g_dirigido.txt' y 'g_nodirigido.txt' para distintos orígenes; registrar orden y niveles (BFS).
- Extender: componentes (no dirigido) o conjuntos alcanzables (dirigido) repitiendo recorridos desde no visitados.
- Plus: detectar ciclo (criterio no dirigido/dirigido).

Cierre

- Comparar órdenes de visita de DFS y BFS.
- Discutir usos: BFS para caminos mínimos no ponderados; DFS para exploración y ciclos/topología (con variantes).

6. Resultados esperados:

- Código 'Graph', 'BFS', 'DFS', 'MainDemo' con ejecución reproducible.
- Tabla con orden de visita y dist[] de BFS por dataset y origen.
- Capturas de salida de consola; breve informe (1–2 págs.)



```
| ~ Archivo cargado: C:\Users\ASUS\Desktop\grafo_prueba\Taller8-Grafos\src\ed\u3\txt\MASTER_PODEROSA_ULTRA_MATRIZ.txt
| ~ Grafo cargado (10000 vertices)

| ====================================== |
          Informacion del grafo
| ====================================== |
| ~ Tipo: Dirigido
| ~ Vertices: 10000

| ------ Lista de adyacencia ------ |
| ~ Vertice: [0] -> []
| ~ Vertice: [1] -> []
| ~ Vertice: [2] -> []
| ~ Vertice: [3] -> []
| ~ Vertice: [4] -> []
| ~ Vertice: [5] -> []
| ~ Vertice: [6] -> []
| ~ Vertice: [7] -> []
| ~ Vertice: [8] -> []
| ~ Vertice: [9] -> []
```

```
| ~ Vertice: [9999] <- []
| ====================================== |

| - Ingrese el vértice de origen (0 a 9999): 555

| ~ Ejecutando BFS desde nodo: 555.

------------------------------------------------------------
          Tabla bfs (dist[] y orden)
------------------------------------------------------------
| Vertice    | Distancia  | Orden          |
------------------------------------------------------------
| 0          | inf        | 555            |
| 1          | inf        | -              |
| 2          | inf        | -              |
| 3          | inf        | -              |
| 4          | inf        | -              |
```

```
| ~ Orden final: [555]
------------------------------------------------------

| ~ Ejecutando DFS desde nodo: 555.

----------------------------------------
           Resultados dfs
----------------------------------------
| ~ Orden de visita (ruta):
|   [555, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,

| ~ Ciclo en el grafo: No detectado
----------------------------------------

| ------------------------------------ |

========================================
          GRAFOS (DFS/BFS)
========================================
| ~ 1. DataSet Grafo NO Dirigido.
| ~ 2. Dataset Grafo Dirigido.
```

Enlace git:

https://github.com/xDylok/Taller8-Grafos

7. Preguntas de Control:

- **¿Por qué BFS entrega distancias mínimas en aristas no ponderadas? ¿Qué cambia si hay pesos?**

  BFS explora el grafo por niveles: primero todos los vecinos a distancia 1, luego a distancia 2, y así sucesivamente. Esto garantiza que la primera vez que "descubrimos" un nodo, lo hayamos hecho por el camino más corto posible en términos de número de aristas. Si el grafo tuviera pesos, BFS fallaría porque un camino con más aristas podría tener un peso total menor; por lo que necesitaríamos usar algoritmos Dijkstra.

- **¿Qué estructuras auxiliares necesita cada algoritmo y por qué?**

  BFS: Utiliza una Cola (Queue), específicamente `ArrayDeque` en nuestra implementación , para gestionar el orden de visita (FIFO) y un vector de distancias y visitados para evitar ciclos infinitos.

  DFS: Requiere una Pila (Stack), ya sea de forma explícita o implícita mediante la recursión , para profundizar en una rama antes de retroceder, además del vector de visitados.

- **¿Cómo obtener componentes conexas con DFS/BFS?**

  Para identificar todos los componentes, no basta con un solo recorrido. Debemos iterar sobre todos los vértices del grafo; si encontramos un vértice que aún no ha sido marcado como "visitado", disparamos un BFS o DFS desde allí. Cada vez que iniciamos un nuevo recorrido desde el bucle principal, hemos encontrado una nueva componente conexa (en grafos no dirigidos) o un nuevo conjunto alcanzable

- **¿Cómo detectas ciclo en no dirigidos con DFS? ¿Y en dirigidos?**

  No dirigidos: Durante el DFS, si encontramos un vecino ya visitado que no es el padre del nodo actual, existe un ciclo.
  Dirigidos: Es más complejo; necesitamos rastrear los nodos que están actualmente en la pila de recursión (antepasados). Si encontramos un arco que apunta a un nodo que todavía está en la pila (un "arco hacia atrás"), hemos detectado un ciclo.

- **¿Cuándo preferirías matriz de adyacencia en lugar de listas?**

  Preferiría la matriz de adyacencia cuando el grafo es muy denso, es decir, el número de aristas es cercano al cuadrado de vértices o cuando necesito verificar de forma constante y rápida si existe una arista específica entre dos nodos. Para grafos dispersos, como los que usamos en el taller, la lista de adyacencia es más eficiente en memoria

8. Conclusiones

- DFS y BFS son algoritmos esenciales en la teoría de grafos y se utilizan ampliamente en diversas aplicaciones. Comprender estos algoritmos y saber cuándo usar cada uno puede mejorar considerablemente habilidades para la resolución de problemas.
- Se debe utilizar **BFS** cuando la prioridad es encontrar la ruta más corta en grafos no ponderados o explorar nodos por cercanía niveles , mientras que **DFS** es la opción preferida para tareas que requieren una exploración profunda, como la detección de ciclos, el ordenamiento topológico o cuando se necesita recorrer laberintos mediante *backtracking*.
- Aprendimos también que el control de nodos "visitados" es el pilar fundamental de ambos algoritmos; sin una correcta aplicacion los resultados de conectividad y detección de ciclos serían erróneos.
- El uso correcto de estructuras auxiliares como Queue en BFS y vectores

de `visitados` es crítico; sin ellos, el algoritmo no podría distinguir entre nodos nuevos y ya procesados, cayendo en bucles infinitos.

9. Evaluación

Criterio	4 – Excelente	3 – Bueno	2 – Básico	1 – Insuficiente	Pts
BFS	Correcto; orden y dist[] válidos; código claro	Funciona con detalles menores	Parcial/errores	No funcional	3.0
DFS	Correcto; rec/iter; ciclo básico	Funciona con detalles menores	Parcial/errores	No funcional	3.0
Pruebas / evidencias	Tablas completas + capturas	Aceptables	Escasas	Nulas	2.0
Informe / conclusiones	Claras: cuándo usar DFS/BFS	Aceptables	Superficiales	Nulas	1.5
Calidad de código	Organizado; README breve	Aceptable	Pobre	Deficiente	0.5

10. Bibliografía

[1] OpenDSA Project, "Graph Traversals: BFS and DFS," Virginia Tech, 2021–2024.

[2] P. W. Bible and L. Moser, An Open Guide to Data Structures and Algorithms. PALNI Open Press, 2023.

[3] S. S. Skiena, The Data Structures and Algorithms Design Manual, 3rd ed., Springer, 2020.

[4]Oracle, "Java SE 17–21 Collections Framework: Queue, Deque, Stack patterns,"
2021–2025.

Elaboración y Aprobación

Elaborado por	Andrés R Navas Castellanos **Docente**	Firmado electrónicamente por: ANDRES ROBERTO NAVAS CASTELLANOS Validar únicamente con FirmaEC
Revisado por **Solo si es realizado en laboratorios**	Luis Sinche **Técnico Docente**	No Aplica
Aprobado por	Edison L Coronel Romero **Director de Carrera**	Firmado electrónicamente por: EDISON LEONARDO CORONEL ROMER Validar únicamente con FirmaEC