Informe del Proyecto Final

Jhon Esteban Gutierrez - 506231099

Junio 2024

Contents

1 Introducción

Este informe describe la implementación de algoritmos de búsqueda en grafos y operaciones básicas en estructuras de datos fundamentales, tales como pilas, colas, árboles binarios y el algoritmo de Dijkstra, usando el lenguaje de programación Java. La importancia de estas estructuras radica en su capacidad para manejar datos de manera eficiente, optimizando el rendimiento de diversas aplicaciones computacionales.

2 Descripción del Código

El código principal implementa dos algoritmos de búsqueda en grafos: BFS (Breadth-First Search) y DFS (Depth-First Search), así como operaciones básicas sobre grafos, tales como inserción, búsqueda y actualización de aristas.

2.1 Estructura del Código

El código se estructura en métodos independientes que realizan operaciones específicas sobre grafos y otros tipos de estructuras de datos. Se presenta un menú interactivo que permite al usuario seleccionar la operación que desea realizar.

2.2 Algoritmo BFS

El algoritmo BFS recorre un grafo por niveles, visitando primero todos los vecinos de un nodo antes de pasar a los vecinos de los vecinos, y así sucesivamente.

```
public static void BFS(){
    int[][] grafo = {
            {0, 1, 1, 0, 0},
            {1, 0, 0, 1, 1},
            {1, 0, 0, 0, 0},
            {0, 1, 0, 0, 0},
            {0, 1, 0, 0, 0}
    };

    bfs(grafo, 0);


}

public static void bfs(int[][] grafo, int nodoInicial) {
```

```
        int numNodos = grafo.length;
        boolean[] visitado = new boolean[numNodos];

        Queue<Integer> cola = new LinkedList<>();
        visitado[nodoInicial] = true;
        cola.add(nodoInicial);

        System.out.println("\nRecorrido BFS empezando desde el
            v rtice " + nodoInicial + ":");
        while (!cola.isEmpty()) {
            int nodoActual = cola.poll();
            System.out.print(nodoActual + " ");

            for (int i = 0; i < numNodos; i++) {
                if (grafo[nodoActual][i] == 1 && !visitado[i]) {
                    visitado[i] = true;
                    cola.add(i);
                }
            }
        }
}
```

Listing 1: BFS

La complejidad temporal del algoritmo BFS es $O(V + E)$, donde V es el número de vértices y E es el número de aristas del grafo.

2.3 Algoritmo DFS

El algoritmo DFS recorre un grafo en profundidad, siguiendo un camino hasta el final antes de retroceder y explorar otros caminos.

```
public static void DFS() {
    int[][] grafo = {
            {0, 1, 1, 0, 0},
            {1, 0, 0, 1, 1},
            {1, 0, 0, 0, 0},
            {0, 1, 0, 0, 0},
            {0, 1, 0, 0, 0}
    };

    dfs(grafo, 0, new boolean[grafo.length]);

    // Men  interactivo omitido por brevedad
}

public static void dfs(int[][] grafo, int nodo, boolean[]
    visitado) {
    visitado[nodo] = true;
```

```
        System.out.print(nodo + "-");

        for (int i = 0; i < grafo.length; i++) {
            if (grafo[nodo][i] == 1 && !visitado[i]) {
                dfs(grafo, i, visitado);
            }
        }
        System.out.println();
    }
```

Listing 2: DFS

La complejidad temporal del algoritmo DFS es $O(V+E)$, similar al BFS, donde V es el número de vértices y E es el número de aristas del grafo.

2.4 Operaciones Básicas en Grafos

Los métodos para insertar, buscar y actualizar aristas en el grafo se presentan a continuación.

```
public static void insertarArista(int[][] grafo, int nodoInicial
    , int nodoFinal) {
    grafo[nodoInicial][nodoFinal] = 1;
    grafo[nodoFinal][nodoInicial] = 1; // Si es un grafo no
        dirigido
}

public static boolean buscarArista(int[][] grafo, int
    nodoInicial, int nodoFinal) {
    return grafo[nodoInicial][nodoFinal] == 1;
}

public static void actualizarArista(int[][] grafo, int
    nodoInicial, int nodoFinal, int nuevoValor) {
    grafo[nodoInicial][nodoFinal] = nuevoValor;
    grafo[nodoFinal][nodoInicial] = nuevoValor; // Si es un
        grafo no dirigido
}
```

Listing 3: Operaciones en Grafos

La complejidad de las operaciones básicas en grafos es la siguiente: - Inserción de arista: $O(1)$ - Búsqueda de arista: $O(1)$ - Actualización de arista: $O(1)$

3 Importancia de las Estructuras de Datos Utilizadas

Las estructuras de datos utilizadas en este proyecto, como grafos, pilas, colas y árboles, son fundamentales en la informática. Cada una tiene características únicas que las hacen adecuadas para diferentes tipos de problemas.

3.1 Grafos

Los grafos son cruciales para modelar relaciones entre objetos. Se utilizan en redes de comunicación, sistemas de recomendación y muchas otras aplicaciones. Los algoritmos BFS y DFS son esenciales para explorar estas relaciones eficientemente.

3.2 Pilas y Colas

Las pilas y colas son estructuras de datos lineales que permiten operaciones de inserción y eliminación en un orden específico. Las pilas siguen el principio LIFO (Last In, First Out), mientras que las colas siguen el principio FIFO (First In, First Out). Estas estructuras son ampliamente utilizadas en la gestión de memoria, procesamiento de tareas y más.

3.3 Árboles

Los árboles son estructuras jerárquicas que se utilizan para organizar datos de manera que permitan búsquedas, inserciones y eliminaciones rápidas. Los árboles binarios, en particular, son la base de muchas estructuras de datos más complejas, como los árboles AVL y los árboles B+.

4 Complejidad Algorítmica

La complejidad algorítmica de las estructuras de datos y algoritmos implementados es un aspecto crucial para entender su eficiencia y rendimiento en diferentes escenarios.

4.1 Grafos

- BFS y DFS: $O(V + E)$ - Inserción de arista: $O(1)$ - Búsqueda de arista: $O(1)$ - Actualización de arista: $O(1)$

4.2 Pilas y Colas

- Operaciones en pilas (push, pop): $O(1)$ - Operaciones en colas (enqueue, dequeue): $O(1)$

4.3 Árboles

- Inserción en árboles binarios: $O(\log n)$ en promedio, $O(n)$ en el peor caso - Búsqueda en árboles binarios: $O(\log n)$ en promedio, $O(n)$ en el peor caso - Eliminación en árboles binarios: $O(\log n)$ en promedio, $O(n)$ en el peor caso

5 Conclusión

En conclusión, las estructuras de datos son componentes fundamentales en la programación y la informática. La implementación de algoritmos eficientes para manipular estas estructuras puede mejorar significativamente el rendimiento y la capacidad de resolución de problemas en diversas aplicaciones. Este proyecto demuestra la importancia y la utilidad de los grafos, pilas, colas y árboles en el desarrollo de soluciones computacionales efectivas.