Nama : Dini Auliya Fauziah

Kelas : 47-02

Nim : 607062300015

TUGAS JURNAL 10 BFS DAN DFS

1. Berikut adalah pengerjaan untuk BFS:

   Alur BFS :

```java
import java.util.LinkedList;
import java.util.Queue;

public class bfs {
    private int nodes;
    private LinkedList<Integer> adj[];
    private Queue<Integer> vertexQueue;

    // untuk pembuatan list adjacency dan inisialisasi vertexQueue
    public bfs(int vertex) {
        nodes = vertex;
        // untuk menginisialisasi linkedlist untuk menyimopan tetangga
        adj = new LinkedList[nodes];
        for (int i = 0; i < vertex; i++) {
            adj[i] = new LinkedList<>();
        }
        vertexQueue = new LinkedList<Integer>();
    }

    // List Adjacency, penambahan edge
    public void insertEdge(char source, char dest) {
        adj[source - 'A'].add(dest - 'A');
    }

    // untuk melihat hasil List Adjacency
    public void getAdj() {
        for (LinkedList<Integer> linkedList : adj) {
            System.out.println(linkedList);
        }
    }

    // Penelusuran BFS
    public void bfs(char sumber) {
        // membuat queue untuk verteks yang sudah dikunjungi
        boolean traversalOrder[] = new boolean[nodes];
        // nilai awal verteks yang dikunjungi
        int vertexDikunjungi = 0;
        traversalOrder[sumber - 'A'] = true;// vertex sumber di-visit dan dimasukkan kedalam queue
        vertexQueue.add(sumber - 'A'); // menambahkan simpul awal ke dalam antrian

        while (!vertexQueue.isEmpty()) {
```

```
        while (!vertexQueue.isEmpty()) {
            // ambil dan hapus vertex pertama
            vertexDikunjungi = vertexQueue.poll();
            // print vertex pertama
            System.out.print((char) (vertexDikunjungi + 'A') + " ");
            // cek tetangga dari vertex awal
            for (int tetangga : adj[vertexDikunjungi]) {
                // jika tetangga belum dikunjungi, kunjungi tetangga
                if (!traversalOrder[tetangga]) {
                    // untuk menandai bahwa simpul tetangga telah dikunjungi
                    traversalOrder[tetangga] = true;
                    // untuk menambahkan tetangga yang belum dikunjungi ke dalam antrian
                    vertexQueue.add(tetangga);
                }
            }
        }
    }
}
```

BFS Main :

```
public class bfsMain {
    Run | Debug
    public static void main(String[] args) {
        // untuk membuat objek bfs dengan 9 simpul
        bfs graf = new bfs(vertex:9);
        graf.insertEdge(source:'A', dest:'B');
        graf.insertEdge(source:'A', dest:'D');
        graf.insertEdge(source:'A', dest:'E');
        graf.insertEdge(source:'B', dest:'E');
        graf.insertEdge(source:'D', dest:'G');
        graf.insertEdge(source:'E', dest:'H');
        graf.insertEdge(source:'E', dest:'F');
        graf.insertEdge(source:'G', dest:'H');
        graf.insertEdge(source:'H', dest:'I');
        graf.insertEdge(source:'F', dest:'H');
        graf.insertEdge(source:'F', dest:'C');
        graf.insertEdge(source:'I', dest:'F');
        graf.insertEdge(source:'C', dest:'B');
        System.out.println(x:"Berikut output dari BFS : ");
        graf.bfs(sumber:'A');
    }
}
```

Output BFS :

```
Berikut output dari BFS :
A B D E G H F I C
PS C:\Semester 2\ISD\Jurnal10>
```

2. Berikut adalah pengerjaan dari DFS :

Alur DFS :

```java
import java.util.Iterator;
import java.util.LinkedList;
import java.util.Stack;

public class dfs {
    // untuk menyimpan jumlah simpul di graf
    private int verteks;
    // untuk menyimpan daftar tetangga dari tetangga
    private LinkedList<Integer> adj[];

    // array penanda visited
    private boolean visited[];

    // Constructor
    @SuppressWarnings("unchecked")
    public dfs(int v) {
        verteks = v;
        // inisialisasi linkedlist untuk menyimoan tetangga
        adj = new LinkedList[verteks];
        for (int i = 0; i < v; ++i)
            adj[i] = new LinkedList<Integer>();

        visited = new boolean[verteks];
        // untuk mengatur semua simpul sebagai yang belum dikunjungi
        for (int i = 0; i < verteks; i++) {
            visited[i] = false;
        }
    }

    public void addEdge(char source, char dest) {
        adj[source - 'A'].add(dest - 'A');
    }

    public void dfs(char sourceVertex) {
        // membuat stack untuk dfs
        Stack<Integer> stack = new Stack<>();

        // untuk menambakan simpul awal ke dalam stack
        stack.push(sourceVertex - 'A');

        while (!stack.isEmpty()) {
```

```java
        while (!stack.isEmpty()) {
            // untuk mengambil simpul dari atas stack
            int source = stack.pop();

            if (!visited[source]) {
                // akan menandai simpul sebagai yang sedang dikunjungi
                visited[source] = true;
                System.out.print((char) (source + 'A') + " ");

                // Mendapatkan tetangga dari simpul yang diambil dari stack
                // Jika tetangga belum dikunjungi, maka ditambahkan ke dalam stack
                Iterator<Integer> itr = adj[source].iterator();
                while (itr.hasNext()) {
                    int v = itr.next();
                    if (!visited[v]) {
                        stack.push(v);
                    }
                }
            }
        }
    }
}
```

DFS Main :

```java
public class dfsMain {
    Run | Debug
    public static void main(String[] args) {
        // untuk membuat objek DFSTraversal denhan 9 simpul
        dfs g = new dfs(v:9);
        g.addEdge(source:'A', dest:'B');
        g.addEdge(source:'B', dest:'E');
        g.addEdge(source:'E', dest:'H');
        g.addEdge(source:'H', dest:'I');
        g.addEdge(source:'I', dest:'F');
        g.addEdge(source:'F', dest:'C');
        g.addEdge(source:'C', dest:'B');
        g.addEdge(source:'C', dest:'D');
        g.addEdge(source:'D', dest:'G');
        System.out.println(x:"Berikut output dari DFS :");
        g.dfs(sourceVertex:'A');
    }
}
```

Output DFS :

```
Berikut output dari DFS :
A B E H I F C D G
PS C:\Semester 2\ISD\Jurnal10>
```