

Tugas 2

Algoritma dan Struktur Data

Teknik Informatika Kelas C

Nama:

Zhafran Rama Azmi (215150207111025)

Dosen:

Putra Pandu Adikara, S.Kom., M.Kom.



**Program Studi Teknik Informatika
Jurusan Teknik Informatika
Universitas Brawijaya**

2022

1. Kode

twitty.java

```
import java.util.Scanner;
// Zhafran Rama Azmi (215150207111025)
public class twitty {
    public static void main(String[] args) {
        Graph twittyGraph = new Graph();
        Scanner in = new Scanner(System.in);

        int x = in.nextInt();
        in.nextLine();
        int y = in.nextInt();
        in.nextLine();

        for (int i = 0; i < x; i++) {
            String[] insert = new String[4];
            insert = in.nextLine().split(" ");
            twittyGraph.addVertex(insert[0], insert[1], insert[2],
insert[3]);
        }

        for (int i = 0; i < y; i++) {
            String[] insert = new String[2];
            insert = in.nextLine().split(" ");
            twittyGraph.connect(insert[0], insert[1]);
        }

        while (in.hasNextLine()) {
            String input[] = in.nextLine().split(" ");

            if (input[0].equals("connect")) {
                twittyGraph.connect(input[1], input[2]);
                System.out.println("connect " + input[1] + " " +
input[2] + " success");
            }
            if (input[0].equals("insert")) {
                twittyGraph.addVertex(input[1], input[2], input[3],
input[4]);
                System.out.println(input[1] + " inserted");
            }
            if (input[0].equals("mostfollowed")) {
                System.out.println(twittyGraph.getNodeMostFoll().nama);
            }
            if (input[0].equals("minrt")) {
                System.out.println();
            }
        }
    }
}
```

```
        if (input[0].equals("numgroup")) {
            System.out.println();
        }
        if (input[0].equals("grouptopic")) {
            System.out.println();
        }
    }
}

class Graph {
    vertexList vertexList = new vertexList();

    void addVertex(String nama, String minat1, String minat2, String
minat3) {
        vertexList.add(nama, minat1, minat2, minat3);
    }

    void connect(String asal, String tujuan) {
        NodeVertex tmp1 = vertexList.findByName(asal);
        NodeVertex tmp2 = vertexList.findByName(tujuan);
        tmp1.adj.addLast(tmp2);
        tmp2.follower++;
    }

    int getMaxFollower() {
        NodeVertex tmp = vertexList.head;
        int max = 0;
        while (tmp != null) {
            if (tmp.follower > max) {
                max = tmp.follower;
            }
            tmp = tmp.next;
        }
        return max;
    }

    NodeVertex getNodeMostFoll() {
        int x = getMaxFollower();
        NodeVertex tmp = vertexList.head;
        while (tmp != null) {
            if (tmp.follower == x) {
                return tmp;
            }
            tmp = tmp.next;
        }
    }
}
```

```

        return null;
    }

    // int shortestPath(String asal, String tujuan, int counter) {
    //     NodeVertex tmp1 = vertexList.head;
    //     NodeVertex tmp2 = vertexList.head;

    //     int adjCounter;
    //     int pathCounter;

    //     while (tmp1 != vertexList.findByName(asal) && tmp1 != null) {
    //         tmp1 = tmp1.next;
    //     }
    //     while (tmp2 != vertexList.findByName(tujuan) && tmp2 != null)
    {
        //         tmp2 = tmp2.next;
        //     }

        //     Node<NodeVertex> adjtmp1 = tmp1.adj.head;

        //     while(adjtmp1 != null) {
        //         if (adjtmp1.data != tmp2) {
        //             adjtmp1 = adjtmp1.next;
        //             if () {

        //                 }
        //             } else {
        //                 counter++;
        //                 return counter;
        //             }
        //         }
        //         if (tmp1.next==null) {
        //             return -1;
        //         }
        //     }
    }

class vertexList {
    NodeVertex head;
    int size;

    void add(String nama, String minat1, String minat2, String minat3) {
        NodeVertex p = new NodeVertex(nama, minat1, minat2, minat3);
        if (head == null) {
            head = p;
            return;
        }
    }
}

```

```
    }
    NodeVertex temp = head;
    while (temp.next != null) {
        temp = temp.next;
    }
    temp.next = p;
    size++;
}

NodeVertex findByName(String nama) {
    NodeVertex tmp = head;
    while (tmp.nama != nama && tmp.next != null) {
        tmp = tmp.next;
    }
    return tmp;
}

}

class NodeVertex {
    String nama;
    String minat[] = new String[3];
    NodeVertex next;
    singly<NodeVertex> adj = new singly<>();
    int follower=0;

    NodeVertex(String nama, String minat1, String minat2, String minat3)
    {
        this.nama = nama;
        minat[0]=minat1;
        minat[1]=minat2;
        minat[2]=minat3;
    }
}

class Node<T> {
    T data;
    Node<T> next;
    Node() {}
    Node(T newData) {
        data = newData;
    }
    Node (T newData, Node<T> newNext) {
        next = newNext;
        data = newData;
    }
}
```

```
}  
  
}  
  
class singly<T> {  
    Node<T> head,tail;  
    int size=0;  
    singly() {  
        head = null;  
        tail =null;  
    }  
    boolean isEmpty() {  
        return size == 0 || head == null;  
    }  
    void addFirst(T x) {  
        Node<T> p = new Node<>();  
        p.data=x;  
        p.next = head;  
        head = p;  
        size++;  
    }  
    void addLast(T x) {  
        Node<T> p = new Node<>(x);  
        if (head == null) {  
            head = p;  
            return;  
        }  
        Node<T> temp = head;  
        while (temp.next != null) {  
            temp = temp.next;  
        }  
        temp.next = p;  
        size++;  
    }  
    int length() {  
        return size;  
    }  
    void insert(T data, int position) {  
        Node<T> node = new Node<>(data);  
        if (position > size) {  
            addLast(data);  
            return;  
        }  
        else if (position == 1) {  
            addFirst(data);  
        } else {
```

```
Node<T> previous = head;
int count = 1;
while (count < position-1) {
    previous = previous.next;
    count++;
}
Node<T> current = previous.next;
node.next = current;
previous.next = node;
}
size++;
}
```

Refleksi Diri:

Traversal pada suatu graph cukup sulit, perlu saya teliti lebih banyak lagi.

2. Output

```
→ cat input | java twitty.java
rozi
dolly inserted
wini inserted

bolly inserted
biti inserted
tini inserted
connect dolly bolly success
connect bolly dolly success
connect biti tini success
connect tini wini success
connect biti wini success
connect wini biti success

rozi
```

Refleksi Diri: