Modul

7

Graph

Tujuan

Mahasiswa diharapkan dapat menerapkan penggunaan graph pada suatu kasus. Secara khusus program yang mengandung instruksi-instruksi yang menerapkan bagian dari graph. Mahasiswa juga dapat menggunakan node dan edges pada penggunaan graph sesuai dengan kasus yang akan diselesaikan. Di akhir praktikum ini mahasiswa dapat:

- Menjelaskan konsep dasar graph
- Memahami bagaimana penerapan node pada graph
- Memahami bagaimana penerapan edges pada graph

Pendahuluan

Praktikum ini mengasumsikan bahwa mahasiswa telah dapat mengoperasikan Netbeans. Mahasiswa juga diharapkan sudah memahami dengan baik materi-materi yang telah diberikan sebelumnya (pemrograman dasar I). Agar mahasiswa dapat mencapai tujuan dalam pertemuan praktikum yang pertama ini, mahasiswa harus memiliki pengetahuan (minimal telah membaca) penggunaan konsep dasar graph. Bagian dari graph yaitu: node dan edges

Proses

Praktikan membaca buku/diktat yang dilakukan selama 20 menit termasuk membuat ringkasan penting, kemudian berdiskusi sesuai dengan panduan aktifitas yang dilakukan selama 20 menit, sedangkan waktu untuk mengerjakan *project* (membuat program/*coding*) secara individual harus diselesaikan di dalam laboratorium dalam waktu 30 menit. Berikutnya untuk sesi latihan, ada soal tentang pengembangan program yang juga harus diselesaikan di laboratorium dalam waktu 50 menit. Terakhir adalah bagian tugas, yaitu *project* yang dikerjakan dirumah dan wajib dikumpulkan pada pertemuan berikutnya.

Aktifitas

1. Mahasiswa membaca buku ajar (jika ada)/diktat kuliah/materi dari sumber lain tentang konsep dasar graph. Temukan bagian penting dalam topik class dan object ini, kemudian tulis sebagai ringkasan hasil belajar.

2. Mahasiswa berdiskusi tentang definisi graph, tujuan adanya node dan edges sebagai bahan diskusi

a. Pada package sebelumnya yaitu bab Tree. Tambahkan class baru dengan nama GraphNode dan tulislah seperti di bawah ini.

```
    class GraphNode{
    int data;

    public GraphNode(int new_data){
        this.data = new_data;
    }
}
```

b. Lengkapilah bagian yang kosong pada class GraphEdge di bawah ini sesuai dengan perintah / command yang tertera.

```
class GraphEdge{
    GraphNode src;
    GraphNode dst;
    double distance;

/*  set this.src into new_src
     * set this.dst into new_dst
     * set this.distance into new_distance
     */

    public GraphEdge(GraphNode new_src, GraphNode new_dst, double
new_distance){
                ……….
                ……….
 ……..……..
    }
}
```

c. Lengkapilah bagian yang kosong pada class Graph di bawah ini sesuai dengan perintah / command yang tertera.

```
    import java.util.ArrayList;

class Graph{
    ArrayList<GraphNode> nodes;
    ArrayList<GraphEdge> edges;


/*  set this.nodes into new Arraylist<GraphNode>
set this.edges into new Arraylist<GraphEdge>
     */
```

```
    public Graph(){
        ……
         ……..
    }

    void add_node(GraphNode new_node){
        this.nodes.add(new_node);
    }

    void add_edge(GraphEdge new_edge){
        this.edges.add(new_edge);
    }

    void remove_node(GraphNode deleted_node){
        this.nodes.remove(deleted_node);
        int i = 0;
        while(i<this.edges.size()){
            GraphEdge edge = edges.get(i);
            if(edge.src == deleted_node || edge.dst ==
deleted_node){
                this.edges.remove(edge);
            }else{
                i++;
            }
        }
    }
void remove_edge(GraphEdge deleted_edge){
        this.edges.remove(deleted_edge);
    }

    ArrayList<GraphEdge> get_edges_by_source_node(GraphNode node){
        ArrayList<GraphEdge> node_edges = new
ArrayList<GraphEdge>();
        for(int i=0; i<this.edges.size(); i++){
            GraphEdge edge = this.edges.get(i);
            if(edge.src == node || edge.dst == node){
                node_edges.add(edge);
            }
        }
        return node_edges;
    }
GraphNode get_node_by_data(int data){
        for(int i=0; i<this.nodes.size(); i++){
            GraphNode node = this.nodes.get(i);
            if(node.data == data){
                return node;
            }
        }
        return null;
    }


    Tree to_tree(int root_data){
        TreeNode first_tree_node = new TreeNode(root_data);
        first_tree_node =
this.completing_tree_node(first_tree_node);
        Tree t = new Tree(first_tree_node);
        return t;
    }
//to be continued next page
```

```
TreeNode completing_tree_node(TreeNode tree_node){
       int data = tree_node.data;
       GraphNode graph_node = this.get_node_by_data(data);
       ArrayList<GraphEdge> edges =
this.get_edges_by_source_node(graph_node);
       for(int i=0; i<edges.size(); i++){ // for all edges
           GraphEdge edge = edges.get(i);
           if(edge.src == graph_node){ // if edge.src == current
graph_node representation of tree_node
               int new_data = edge.dst.data; // get new data
               boolean should_add_new_data = true; // assuming
should add new data unless new data is already in the path of the
tree
               TreeNode current_tree_node = tree_node;
               while(current_tree_node != null){
                   if(current_tree_node.data == new_data){
                       should_add_new_data = false;
                       break;
                   }
                   current_tree_node = current_tree_node.parent;
               }
               if(should_add_new_data){
                   TreeNode new_tree_node = new TreeNode(new_data);
                   tree_node.add_child(new_tree_node,
edge.distance);
                   int last_index = tree_node.children.size()-1;
                   tree_node.children.set(last_index,
this.completing_tree_node(new_tree_node));
               }
           }
       }
       return tree_node;
    }
}
```

d. Simpan class GraphNode.java dan Graph.java pada satu package yang sama. Selanjutnya buat class baru pada package yang sama ketikkan class Test.java di bawah ini dan jalankan.

```
    public class Test{
    public static void main(String Args[]){
 Graph g = new Graph();
       GraphNode[] graph_node_list = {
           new GraphNode(0),
           new GraphNode(1),
           new GraphNode(2),
           new GraphNode(3),
           new GraphNode(4),
       };
       for(GraphNode graph_node : graph_node_list){
           g.add_node(graph_node);
       }
       int[][] path_list = {
           {0,1, 1},
           {0,2, 1},
           {1,3, 1},
```

```
            {2,3, 1},
            {3,4, 2}
        };
        for(int[] path : path_list){
            GraphNode first_node = graph_node_list[path[0]];
            GraphNode second_node = graph_node_list[path[1]];
            double distance = path[2];
            g.add_edge(new GraphEdge(first_node, second_node,
distance));
            g.add_edge(new GraphEdge(second_node, first_node,
distance));
        }
        g.to_tree(0).print();
    }}
```

e. Setelah dijalankan akan muncul output seperti di bawah ini :

```
0 distance from parent : 0.0 distance from initial node : 0.0
1 distance from parent : 1.0 distance from initial node : 1.0
3 distance from parent : 1.0 distance from initial node : 2.0
2 distance from parent : 1.0 distance from initial node : 3.0
4 distance from parent : 2.0 distance from initial node : 4.0
2 distance from parent : 1.0 distance from initial node : 1.0
3 distance from parent : 1.0 distance from initial node : 2.0
1 distance from parent : 1.0 distance from initial node : 3.0
4 distance from parent : 2.0 distance from initial node : 4.0
```

Kesimpulan

Buatlah interface dari program di atas!

Penutup

Tugas

Buatlah sebuah program yang menerapkan konsep graph!