JOBSHEET XII
Graph

1. Tujuan Praktikum

Setelah melakukan praktikum ini, mahasiswa mampu:

1. memahami model graph;
2. membuat dan mendeklarasikan struktur algoritma graph;
3. menerapkan algoritma dasar graph dalam beberapa studi kasus.

2. Praktikum

2.1 Implementasi Graph menggunakan Linked List

2.1.1 Tahapan Percobaan

Waktu percobaan (30 menit)

Pada percobaan ini akan diimplementasikan Graph menggunakan Linked Lists untuk merepresentasikan graph adjacency. Silakan lakukan langkah-langkah praktikum sebagai berikut.

1. Buatlah class **Node**, dan class **Linked Lists** sesuai dengan praktikum **Double Linked Lists.**

Graph
vertex: int LinkedList: List right: Node
addEdge(source: int, destination: int): void degree(source: int): void removeEdge(source: int, destination: int): void removeAllEdges() printGraph()

2. Tambahkan class **Graph** yang akan menyimpan method-method dalam graph dan juga method main().

```
1   public class Graph {
2
3   }
```

3. Di dalam class **Graph**, tambahkan atribut **vertex** bertipe integer dan **list[]** bertipe LinkedList.

```
1    public class Graph {
2        int vertex;
3        LinkedList list[];
```

4. Tambahkan konstruktor default untuk menginisialisasi variabel vertex dan menambahkan perulangan untuk jumlah vertex sesuai dengan jumlah length array yang telah ditentukan.

```
5    public Graph(int vertex) {
6        this.vertex = vertex;
7        list = new LinkedList[vertex];
8        for (int i = 0; i <vertex ; i++) {
9            list[i] = new LinkedList();
10       }
11   }
```

5. Tambahkan method **addEdge()**. Jika yang akan dibuat adalah graph berarah, maka yang dijalankan hanya baris pertama saja. Jika graph tidak berarah yang dijalankan semua baris pada method **addEdge()**.

```
13   public void addEdge(int source, int destination){
14       //add edge
15       list[source].addFirst(destination);
16
17       //add back edge (for undirected)
18       list[destination].addFirst(source);
19   }
```

6. Tambahkan method **degree()** untuk menampilkan jumlah derajat lintasan pada suatu vertex. Di dalam metode ini juga dibedakan manakah statement yang digunakan untuk graph berarah atau graph tidak berarah. Eksekusi hanya sesuai kebutuhan saja.

```
public void degree(int source) throws Exception{
    //degree undirected graph
    System.out.println("degree vertex "+source +" : "+list[source].size());

    //degree directed graph
        //inDegree
    int k,totalIn = 0, totalOut = 0;
    for (int i = 0; i <vertex ; i++) {
    for (int j = 0; j < list[i].size(); j++) {
         if(list[i].get(j)==source)
                ++totalIn;
    }
        //outDegree
    for (k = 0; k < list[source].size(); k++) {
        list[source].get(k);
    }
    totalOut = k;
    }
    System.out.println("Indegree dari vertex "+ source +" : "+totalIn);
    System.out.println("Outdegree dari vertex "+ source +" : "+totalOut);
    System.out.println("degree vertex "+source +" : "+(totalIn+totalOut));
}
```

7. Tambahkan method **removeEdge()**. Method ini akan menghapus lintasan ada suatu graph. Oleh karena itu, dibutuhkan 2 parameter untuk menghapus lintasan yaitu source dan destination.

```
44      public void removeEdge(int source, int destination) throws Exception{
45          for (int i = 0; i <vertex ; i++) {
46              if(i==destination){
47                  list[source].remove(destination);
48              }
49          }
50      }
```

8. Tambahkan method **removeAllEdges()** untuk menghapus semua vertex yang ada di dalam graph.

```
52      public void removeAllEdges() {
53          for (int i = 0; i <vertex ; i++) {
54          list[i].clear();
55          }
56          System.out.println("Graph berhasil dikosongkan");
57      }
```

9. Tambahkan method **printGraph()** untuk mencatak graph ter-*update.*

```
public void printGraph() throws Exception{
    for (int i = 0; i <vertex ; i++) {
        if(list[i].size()>0) {
            System.out.print("Vertex " + i + " terhubung dengan: ");
            for (int j = 0; j < list[i].size(); j++) {
                System.out.print(list[i].get(j) + " ");
            }
            System.out.println("");
        }
    }
    System.out.println(" ");
}
```

10. Compile dan jalankan method **main()** dalam class **Graph** untuk menambahkan beberapa edge pada graph, kemudian tampilkan. Setelah itu keluarkan hasilnya menggunakan pemanggilan method main(). **Keterangan:** degree harus disesuaikan dengan jenis graph yang telah dibuat (directed/undirected).

```
72      public static void main(String[] args) throws Exception {
73          Graph graph = new Graph(6);
74          graph.addEdge(0, 1);
75          graph.addEdge(0, 4);
76          graph.addEdge(1, 2);
77          graph.addEdge(1, 3);
78          graph.addEdge(1, 4);
79          graph.addEdge(2, 3);
80          graph.addEdge(3, 4);
81          graph.addEdge(3, 0);
82          graph.printGraph();
83          graph.degree(2);
84
85      }
```

11. Amati hasil running tersebut.
12. Tambahkan pemanggilan method **removeEdge()** sesuai potongan code di bawah ini pada method main(). Kemudian tampilkan graph tersebut.

```
89              graph.removeEdge(1, 2);
90              graph.printGraph();
```

13. Amati hasil running tersebut.
14. Uji coba penghapusan lintasan yang lain! Amati hasilnya!

2.1.2 Verifikasi Hasil Percobaan

Verifikasi hasil kompilasi kode program Anda dengan gambar berikut ini.

Hasil running pada langkah ke-11

```
--- exec-maven-plugin:1.5.0:exec (default-cli)
Vertex 0 terhubung dengan: 3 4 1
Vertex 1 terhubung dengan: 4 3 2 0
Vertex 2 terhubung dengan: 3 1
Vertex 3 terhubung dengan: 0 4 2 1
Vertex 4 terhubung dengan: 3 1 0

degree vertex 2 : 2
Indegree dari vertex 2 : 2
Outdegree dari vertex 2 : 2
degree vertex 2 : 4
------------------------------------------------
BUILD SUCCESS
------------------------------------------------
```

Hasil running pada langkah ke-13

```
--- exec-maven-plugin:1.5.0:exec (default-cli)
Vertex 0 terhubung dengan: 3 4 1
Vertex 1 terhubung dengan: 4 3 2 0
Vertex 2 terhubung dengan: 3 1
Vertex 3 terhubung dengan: 0 4 2 1
Vertex 4 terhubung dengan: 3 1 0

degree vertex 2 : 2
Indegree dari vertex 2 : 2
Outdegree dari vertex 2 : 2
degree vertex 2 : 4
Vertex 0 terhubung dengan: 3 4 1
Vertex 1 terhubung dengan: 4 3 0
Vertex 2 terhubung dengan: 3 1
Vertex 3 terhubung dengan: 0 4 2 1
Vertex 4 terhubung dengan: 3 1 0

------------------------------------------------
BUILD SUCCESS
------------------------------------------------
```

2.1.3 Pertanyaan Percobaan

1. Sebutkan beberapa jenis (minimal 3) algoritma yang menggunakan dasar Graph, dan apakah kegunaan algoritma-algoritma tersebut?
2. Pada class Graph terdapat array bertipe LinkedList, yaitu LinkedList list[]. Apakah tujuan pembuatan variabel tersebut ?
3. Apakah alasan pemanggilan method **addFirst()** untuk menambahkan data, bukan method add jenis lain pada linked list ketika digunakan pada method addEdge pada **class Graph?**
4. Bagaimana cara mendeteksi prev pointer pada saat akan melakukan penghapusan suatu edge pada graph ?
5. Kenapa pada praktikum 2.1.1 langkah ke-12 untuk menghapus path yang bukan merupakan lintasan pertama kali menghasilkan output yang salah ? Bagaimana solusinya ?

```
89          graph.removeEdge(1, 3);
90          graph.printGraph();
```

2.2 Implementasi Graph menggunakan Matriks

Kegiatan praktikum 2 merupakan implementasi Graph dengan Matriks. Silakan lakukan langkah-langkah percobaan praktikum berikut ini, kemudian verifikasi hasilnya. Setelah itu jawablah pertanyaan terkait percobaan yang telah Anda lakukan.

2.2.1 Tahapan Percobaan

Waktu percobaan: 30 menit

Pada praktikum 2.2 ini akan diimplementasikan Graph menggunakan matriks untuk merepresentasikan graph adjacency. Silakan lakukan langkah-langkah praktikum sebagai berikut.

1. Uji coba graph bagian 2.2 menggunakan array 2 dimensi sebagai representasi graph. Buatlah class **graphArray** yang didalamnya terdapat variabel **vertices** dan **array twoD_array**!

```
public class graphArray {

    private final int vertices;
    private final int[][] twoD_array;
```

2. Buatlah konstruktor **graphArray** sebagai berikut!

```
public graphArray(int v)
{
    vertices = v;
    twoD_array = new int[vertices + 1][vertices + 1];
}
```

3. Untuk membuat suatu lintasan maka dibuat method **makeEdge()** sebagai berikut.

```
14      public void makeEdge(int to, int from, int edge)
15      {
16          try
17          {
18              twoD_array[to][from] = edge;
19          }
20          catch (ArrayIndexOutOfBoundsException index)
21          {
22              System.out.println("Vertex tidak ada");
23          }
24      }
```

Untuk menampilkan suatu lintasan diperlukan pembuatan method **getEdge()** berikut.

```
26          public int getEdge(int to, int from)
27          {
28              try
29              {
30                  return twoD_array[to][from];
31              }
32              catch (ArrayIndexOutOfBoundsException index)
33              {
34                  System.out.println("Vertex tidak ada");
35              }
36              return -1;
37          }
```

4. Kemudian buatlah method **main()** seperti berikut ini.

```
public static void main(String args[]) {
    int v, e, count = 1, to = 0, from = 0;
    Scanner sc = new Scanner(System.in);
    graphArray graph;
    try {
        System.out.println("Masukkan jumlah vertices: ");
        v = sc.nextInt();
        System.out.println("Masukkan jumlah edges: ");
        e = sc.nextInt();

        graph = new graphArray(v);

        System.out.println("Masukkan edges: <to> <from>");
        while (count <= e) {
            to = sc.nextInt();
            from = sc.nextInt();

            graph.makeEdge(to, from, 1);
            count++;
        }
        System.out.println("Array 2D sebagai representasi graph sbb: ");
        System.out.print("  ");
        for (int i = 1; i <= v; i++) {
            System.out.print(i + " ");
        }
        System.out.println();

        for (int i = 1; i <= v; i++) {
            System.out.print(i + " ");
            for (int j = 1; j <= v; j++) {
                System.out.print(graph.getEdge(i, j) + " ");
            }
            System.out.println();
        }
    } catch (Exception E) {
        System.out.println("Error. Silakan cek kembali\n" + E.getMessage());
    }
    sc.close();
}
```

5. Jalankan class **graphArray** dan amati hasilnya!

2.2.2 Verifikasi Hasil Percobaan

Verifikasi hasil kompilasi kode program Anda dengan gambar berikut ini.

```
Masukkan jumlah vertices:
5
Masukkan jumlah edges:
6
Masukkan edges: <to> <from>
1 2
1 5
2 3
2 4
2 5
3 4
Array 2D sebagai representasi graph sbb:
  1 2 3 4 5
1 0 1 0 0 1
2 0 0 1 1 1
3 0 0 0 1 0
4 0 0 0 0 0
5 0 0 0 0 0
BUILD SUCCESSFUL (total time: 47 seconds)
```

2.2.3 Pertanyaan Percobaan

1. Apakah perbedaan degree/derajat pada *directed* dan *undirected graph*?
2. Pada implementasi graph menggunakan adjacency matriks. Kenapa jumlah vertices harus ditambahkan dengan 1 pada indeks array berikut?

```
8       public graphArray(int v)
9       {
10          vertices = v;
11          twoD_array = new int[vertices + 1][vertices + 1];
12      }
```

3. Apakah kegunaan method **getEdge()** ?
4. Termasuk jenis graph apakah uji coba pada praktikum 2.2?
5. Mengapa pada method main harus menggunakan *try-catch Exception* ?

3. Tugas Praktikum

1. Ubahlah lintasan pada praktikum 2.1 menjadi inputan!
2. Tambahkan method **graphType** dengan tipe boolean yang akan membedakan *graph* termasuk *directed* atau *undirected graph*. Kemudian update seluruh method yang berelasi dengan method **graphType** tersebut (hanya menjalankan statement sesuai dengan jenis graph) pada praktikum 2.1
3. Modifikasi method **removeEdge()** pada praktikum 2.1 agar tidak menghasilkan output yang salah untuk path selain path pertama kali!
4. Ubahlah tipe data *vertex* pada seluruh graph pada praktikum 2.1 dan 2.2 dari Integer menjadi tipe generic agar dapat menerima semua tipe data dasar Java! Misalnya setiap *vertex* yang awalnya berupa angka 0,1,2,3, dst. selanjutnya ubah menjadi suatu nama daerah seperti Gresik, Bandung, Yogya, Malang, dst.

--- *** ---