LAPORAN PRAKTIKUM
MATA KULIAH ALGORITMA DAN STRUKTUR DATA

Dosen Pengampu : Triana Fatmawati, S.T, M.T

PERTEMUAN - 14 - Graph

Nama : **M. Zidna Billah Faza**

NIM : **2341760030**

Prodi : **D-IV Sistem Informasi Bisnis**

JURUSAN TEKNOLOGI INFORMASI

POLITEKNIK NEGERI MALANG

2024

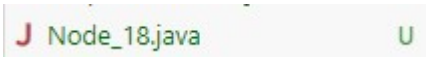
Percobaan 1

Sebuah universitas membuat program untuk memodelkan graf berarah berbobot yang mewakili gedung-gedung dan jarak antar gedung tersebut menggunakan Linked List. Setiap gedung dihubungkan dengan jalan yang memiliki jarak tertentu (dalam meter). Perhatikan class diagram Graph berikut ini.

Graph<NoAbsen>
vertex: int DoubleLinkedList: list[]
addEdge(asal: int, tujuan: int): void degree(asal: int): void removeEdge(asal: int, tujuan: int): void removeAllEdges(): void printGraph(): void

1) Buka text editor. Buat class Node<NoAbsen>.java dan class DoubleLinkedList<NoAbsen>.java sesuai dengan praktikum Double Linked List.

- Class Node_18



Kode program yang terdapat pada class Node belum dapat mengakomodasi kebutuhan pembuatan graf berbobot, sehingga diperlukan sedikit modifikasi. Setelah Anda menyalin kode program dari class Node pada praktikum Double Linked List, tambahkan atribut jarak bertipe int untuk menyimpan bobot graf

```
public class Node_18 {

    int data;
    Node_18 prev, next;
    int jarak;

    Node_18(Node_18 prev, int data, Node_18 next, int jarak) {
        this.prev = prev;
        this.data = data;
        this.next = next;
        this.jarak = jarak;
    }

}
```

- Class DoubleLinkedList_18

Setelah Anda menyalin kode program dari class DoubleLinkedList pada praktikum Double Linked List, lakukan modifikasi pada method addFirst agar dapat menerima parameter jarak dan digunakan saat instansiasi Node.

```
public void AddFirst(int item, int jarak) {
    if (IsEmpty()) {
        head = new Node_18(prev:null, item, head, jarak);
    } else {
        Node_18 newNode = new Node_18(prev:null, item, head, jarak);
        head.prev = newNode;
        head = newNode;
    }
    size++;
}
```

Selanjutnya buat method getJarak (hampir sama seperti method get) yang digunakan untuk mendapatkan nilai jarak edge antara dua node.

```
public int GetJarak(int index) throws Exception {
    if (IsEmpty() || index >= size) {
        throw new Exception(message:"Nilai indeks diluar batas");
    }
    Node_18 tmp = head;
    for (int i = 0; i < index; i++) {
        tmp = tmp.next;
    }
    return tmp.jarak;
}
```

Modifikasi method remove agar dapat melakukan penghapusan edge sesuai dengan node asal dan tujuan pada graf. Pada praktikum Double Linked List, parameter index digunakan untuk menghapus data sesuai posisi pada indeks tertentu, sedangkan pada Graf ini, penghapusan didasarkan pada data node tujuan, sehingga modifikasi kode diperlukan untuk menghindari index out of bound.

```java
public void Remove(int target) throws Exception {
    if (IsEmpty()) {
        throw new Exception(message:"Linked lists masih kosong, tidak dapat dihapus!");
    } else {
        Node_18 current = head;
        while (current != null) {
            if (current.data == target) {
                if (current.prev != null) {
                    current.prev.next = current.next;
                } else {
                    head = current.next;
                }
                if (current.next != null) {
                    current.next.prev = current.prev;
                }
                size--;
                return;
            }
            current = current.next;
        }
        throw new Exception("Node dengan data " + target + " tidak ditemukan");
    }
}
```

2) Buat file baru, beri nama Graph<NoAbsen>.java

3) Lengkapi class Graph dengan atribut yang telah digambarkan di dalam pada class diagram, yang terdiri dari atribut vertex dan DoubleLinkedList.

```java
int vertex;
DoubleLinkedLists_18 list[];
```

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

```java
public Graph_18 (int vertex) {

    this.vertex = vertex;

    list = new DoubleLinkedLists_18[vertex];

    for (int i = 0; i < vertex; i++) {
        list[i] = new DoubleLinkedLists_18();
    }

}
```

5) Tambahkan method addEdge() untuk menghubungkan dua node. Baris kode program berikut digunakan untuk graf berarah (directed).

```
public void AddEdgeDirected(int asal, int tujuan, int jarak) {
    list[asal].AddFirst(tujuan, jarak);
}
```

Apabila graf yang dibuat adalah undirected graph, maka tambahkan kode berikut

```
public void AddEdgeUndirected(int asal, int tujuan, int jarak) {
    list[asal].AddFirst(asal, jarak);
}
```

6) Tambahkan method degree() untuk menampilkan jumlah derajat lintasan pada setiap vertex. Kode program berikut digunakan untuk menghitung degree pada graf berarah

```
public void DegreeDirected(int asal) throws Exception {

    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) == asal) {
                ++totalIn;
            }
        }

        for (k = 0; k < list[asal].Size(); k++) {
            list[asal].Get(k);
        }

        totalOut = k;

    }

    System.out.println("InDegree dari Gedung " + (char) ('A' + asal) + ": " + totalIn);
    System.out.println("OutDegree dari Gedung " + (char) ('A' + asal) + ": " + totalOut);
    System.out.println("Degree dari Gedung " + (char) ('A' + asal) + ": " + (totalOut + totalOut));

}
```

Apabila graf yang dibuat adalah undirected graph, maka cukup gunakan kode berikut.

```
public void DegreeUndirected(int asal) throws Exception {

    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) == asal) {
                ++totalIn;
            }
        }

        for (k = 0; k < list[asal].Size(); k++) {
            list[asal].Get(k);
        }

        totalOut = k;

    }

    System.out.println("Degree dari Gedung " + (char) ('A' + asal) + ": " + list[asal].Size());

}
```

7) Tambahkan method removeEdge() untuk menghapus lintasan pada suatu graph. Penghapusan membutuhkan 2 parameter yaitu node asal dan tujuan.

```
public void RemoveEdge(int asal, int tujuan) throws Exception {
    for (int i = 0; i < vertex; i++) {
        if (i == tujuan) {
            list[asal].Remove(tujuan);
        }
    }
}
```

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

```
public void RemoveAllEdges() {
    for (int i = 0; i < vertex; i++) {
        list[i].Clear();
    }
    System.out.println(x:"Graph berhasil dikosongkan");
}
```

9) Tambahkan method printGraph() untuk mencetak graf

```
public void PrintGraph() throws Exception {
    for (int i = 0; i < vertex; i++) {
        if (list[i].Size() > 0) {
            System.out.println("Degree dari Gedung " + (char) ('A' + i) + " terhubung dengan");
            for (int j = 0; j < list[i].Size(); j++) {
                System.out.println((char) ('A' + list[i].Get(j)) + " (" + list[i].GetJarak(j) + " m), ");
            }
            System.out.println(x:"");
        }
    }
    System.out.println(x:"");
}
```

10) Buat file baru, beri nama GraphMain<NoAbsen>.java

11) Tuliskan struktur dasar bahasa pemrograman Java yang terdiri dari fungsi main

```
Run | Debug
public static void main(String[] args) throws Exception {
```

12) Di dalam fungsi main, lakukan instansiasi object Graph bernama gedung dengan nilai parameternya adalah 6.

```
Graph_18 gedung = new Graph_18(vertex:6);
```

13) Tambahkan beberapa edge pada graf, tampilkan degree salah satu node, kemudian tampilkan grafnya.

```
gedung.AddEdgeDirected(asal:0, tujuan:1, jarak:50);
gedung.AddEdgeDirected(asal:0, tujuan:2, jarak:100);
gedung.AddEdgeDirected(asal:1, tujuan:3, jarak:70);
gedung.AddEdgeDirected(asal:2, tujuan:3, jarak:40);
gedung.AddEdgeDirected(asal:3,tujuan:4, jarak:60);
gedung.AddEdgeDirected(asal:4, tujuan:5, jarak:80);
gedung.DegreeDirected(asal:0);
gedung.PrintGraph();
```

14) Compile dan run program.

```
InDegree dari Gedung A: 0
OutDegree dari Gedung A: 2
Degree dari Gedung A: 4
Degree dari Gedung A terhubung dengan
C (100 m),
B (50 m),

Degree dari Gedung B terhubung dengan
D (70 m),

Degree dari Gedung C terhubung dengan
D (40 m),

Degree dari Gedung D terhubung dengan
E (60 m),

Degree dari Gedung E terhubung dengan
F (80 m),
```

Catatan: Degree harus disesuaikan dengan jenis graf yang digunakan. Pada kasus ini, digunakan directed weighted graph

15) Tambahkan pemanggilan method removeEdge(), kemudian tampilkan kembali graf tersebut.

```
gedung.RemoveEdge(asal:1, tujuan:3);
gedung.PrintGraph();
```

16) Commit dan push kode program ke Github.

https://github.com/zidnafaz/Praktikum-Algoritma-Struktur-Data

17) Compile dan run program serta verifikasi hasil percobaan.

```
Degree dari Gedung A terhubung dengan
C (100 m),
B (50 m),

Degree dari Gedung C terhubung dengan
D (40 m),

Degree dari Gedung D terhubung dengan
E (60 m),

Degree dari Gedung E terhubung dengan
F (80 m),
```

Pertanyaan Percobaan 1

1) Perbaiki kode program Anda apabila terdapat error atau hasil kompilasi kode tidak sesuai!
2) Pada class Graph, terdapat atribut list[] bertipe DoubleLinkedList. Sebutkan tujuan pembuatan variabel tersebut!

Variable tersebut dibuat untuk mengimplementasikan struktur adjacency list yang efisien dan fleksibel dalam menyimpan dan mengelola graph beserta edge dan bobot (jarak) yang terkait.

3) Jelaskan alur kerja dari method removeEdge!

- Lakukan iterasi melalui semua vertex.
- Periksa apakah indeks iterasi (i) sama dengan vertex tujuan.
- Jika sama, panggil method Remove pada adjacency list yang sesuai (list[asal]) untuk menghapus edge ke tujuan.

Method removeEdge menghapus edge yang menghubungkan dua vertex dalam graph dengan mencari dan menghapusnya dari adjacency list yang sesuai.

4) Apakah alasan pemanggilan method addFirst() untuk menambahkan data, bukan method add jenis lain saat digunakan pada method addEdge pada class Graph?

Penggunaan AddFirst dalam method AddEdgeDirected dan AddEdgeUndirected adalah pilihan yang logis untuk memastikan bahwa penambahan edge dilakukan dengan efisien dan sederhana, serta sesuai dengan kebutuhan representasi adjacency list pada graph.

5) Modifikasi kode program sehingga dapat dilakukan pengecekan apakah terdapat jalur antara suatu node dengan node lainnya, seperti contoh berikut (Anda dapat memanfaatkan Scanner).

```
public boolean IsConnected(int node1, int node2) {
    return matriks[node1][node2] != -1;
}
```

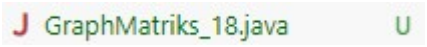
Dengan program tersebut kita dapat mengecek apakah node terhubung

```
Masukkan node ke-1     : 1
Masukkan node ke-2     : 2

Node B C bertetangga
```

Percobaan 2

1) Buat file baru, beri nama GraphMatriks<NoAbsen>.java



2) Lengkapi class GraphMatriks dengan atribut vertex dan matriks

```
int vertex;
int[][] matriks;
```

3) Tambahkan konstruktor default untuk menginisialisasi variabel vertex dan menginstansiasi panjang array dua dimensi yang telah ditentukan.

```
public GraphMatriks_18(int vertex) {
    this.vertex = vertex;
    matriks = new int[vertex][vertex];
}
```

4) Untuk membuat suatu lintasan yang menghubungkan dua node, maka dibuat method makeEdge() sebagai berikut.

```
public void MakeEdge(int asal, int tujuan, int jarak) {
    matriks[asal][tujuan] = jarak;
}
```

5) Tambahkan method removeEdge() untuk menghapus lintasan pada suatu graf.

```
public void RemoveEdge(int asal, int tujuan) {
    matriks[asal][tujuan] = -1;
}
```

6) Tambahkan method printGraph() untuk mencetak graf.

```
public void PrintGraph() {
    for (int i = 0; i < vertex; i++) {
        System.out.print("Gedung " + (char) ('A' + i) + ": ");
        for (int j = 0; j < vertex; j++) {
            if (matriks[i][j] != -1) {
                System.out.print("Gedung " + (char) ('A' + j) + " (" + matriks[i][j] + " m), ");
            }
        }
        System.out.println();
    }
}
```

7) Tambahkan kode berikut pada file GraphMain<NoAbsen>.java yang sudah dibuat pada Percobaan 1.

```
GraphMatriks_18 gdg = new GraphMatriks_18(vertex:4);

gdg.MakeEdge(asal:0, tujuan:1, jarak:50);
gdg.MakeEdge(asal:1, tujuan:0, jarak:60);
gdg.MakeEdge(asal:1, tujuan:2, jarak:70);
gdg.MakeEdge(asal:2, tujuan:1, jarak:80);
gdg.MakeEdge(asal:2, tujuan:3, jarak:40);
gdg.MakeEdge(asal:3, tujuan:0, jarak:90);
gdg.PrintGraph();

System.out.println(x:"\nHasil setelah penghapusan edge\n");

gdg.RemoveEdge(asal:2, tujuan:1);
gdg.PrintGraph();
```

8) Commit dan push kode program ke Github

https://github.com/zidnafaz/Praktikum-Algoritma-Struktur-Data

9) Compile dan run program serta verifikasi hasil percobaan.

```
Gedung A: Gedung A (0m), Gedung B (50m), Gedung C (0m), Gedung D (0m),
Gedung B: Gedung A (60m), Gedung B (0m), Gedung C (70m), Gedung D (0m),
Gedung C: Gedung A (0m), Gedung B (80m), Gedung C (0m), Gedung D (40m),
Gedung D: Gedung A (90m), Gedung B (0m), Gedung C (0m), Gedung D (0m),

Hasil setelah penghapusan edge

Gedung A: Gedung A (0m), Gedung B (50m), Gedung C (0m), Gedung D (0m),
Gedung B: Gedung A (60m), Gedung B (0m), Gedung C (70m), Gedung D (0m),
Gedung C: Gedung A (0m), Gedung B (0m), Gedung C (0m), Gedung D (40m),
Gedung D: Gedung A (90m), Gedung B (0m), Gedung C (0m), Gedung D (0m),
```

Pertanyaan Percobaan 2

1) Perbaiki kode program Anda apabila terdapat error atau hasil kompilasi kode tidak sesuai!
2) Apa jenis graph yang digunakan pada Percobaan 2?

   Graph tersebut adalah graph directed

3) Apa maksud dari dua baris kode berikut?

```
gdg.MakeEdge(asal:1, tujuan:2, jarak:70);
gdg.MakeEdge(asal:2, tujuan:1, jarak:80);
```

   Kode tersebut adalah penambahan edge dimana diinputkan node asal dan tujuan serta jarak antara kedua node tersebut

4) Modifikasi kode program sehingga terdapat method untuk menghitung degree, termasuk inDegree dan outDegree!

```
public int GetInDegree(int node) {
    int inDegree = 0;
    for (int i = 0; i < vertex; i++) {
        if (matriks[i][node] != -1) {
            inDegree++;
        }
    }
    return inDegree;
}

public int GetOutDegree(int node) {
    int outDegree = 0;
    for (int i = 0; i < vertex; i++) {
        if (matriks[node][i] != -1) {
            outDegree++;
        }
    }
    return outDegree;
}
```

```
Masukkan node           : 1
InDegree dari node B    : 5
OutDegree dari node B   : 5
TotalDegree dari node B : 10
```

Tugas Praktikum 1

1) Modifikasi kode program pada class GraphMain sehingga terdapat menu program yang bersifat dinamis, setidaknya terdiri dari:
   - Add Edge
   - Remove Edge
   - Degree
   - Print Graph
   - Cek Edge

   Pengguna dapat memilih menu program melalui input Scanner

```java
int pilihan = 0;

        System.out.print("Masukkan jumlah gedung  : ");
        int vertex = input_18.nextInt();

        GraphMatriks_18 Graph = new GraphMatriks_18(vertex);

        do {

            System.out.println("=================================================");
            System.out.println("                    Graph");
            System.out.println("=================================================");
            System.out.println("\n1. Add Edge");
            System.out.println("2. Remove Edge");
            System.out.println("3. Degree");
            System.out.println("4. Print Graph");
            System.out.println("5. Cek Edge");
            System.out.println("6. Keluar\n");
            System.out.println("=================================================");
            System.out.print("Masukkan Pilihan    : ");
            pilihan = input_18.nextInt();
            System.out.println("=================================================");

            switch (pilihan) {
                case 1:

                    System.out.print("\nMasukkan asal       : ");
                    int asal = input_18.nextInt();
                    System.out.print("Masukkan tujuan     : ");
                    int tujuan = input_18.nextInt();
                    System.out.print("Masukkan jarak      : ");
                    int jarak = input_18.nextInt();
                    System.out.println();

                    Graph.MakeEdge(asal, tujuan, jarak);

                    break;

                case 2:

                    System.out.print("/nMasukkan asal       : ");
                    asal = input_18.nextInt();
                    System.out.print("Masukkan tujuan     : ");
                    tujuan = input_18.nextInt();

                    Graph.RemoveEdge(asal, tujuan);

                    break;

                case 3:

                    System.out.print("\nMasukkan node            : ");
                    int node = input_18.nextInt();

                    System.out.println("InDegree dari node " + (char) ('A' + node) + "     : " + Graph.GetInDegree(node));
                    System.out.println("OutDegree dari node " + (char) ('A' + node) + "    : " + Graph.GetOutDegree(node));
                    System.out.println(
                            "TotalDegree dari node " + (char) ('A' + node) + "  : " + Graph.GetTotalDegree(node));

                    break;

                case 4:

                    Graph.PrintGraph();

                    break;

                case 5:

                    System.out.print("\nMasukkan node ke-1     : ");
                    int node1 = input_18.nextInt();
                    System.out.print("Masukkan node ke-2     : ");
                    int node2 = input_18.nextInt();

                    if (Graph.IsConnected(node1, node2)) {
                        System.out.println("\nNode " + (char) ('A' + node1) + " " + (char) ('A' + node2) + " bertetangga");
                    } else {
                        System.out.println("\nNode " + (char) ('A' + node1) + " " + (char) ('A' + node2) + " tidak bertetangga");

                    }

                    break;

                default:
                    break;
            }

        } while (pilihan != 6);

        input_18.close();
```

```
=================================================
                    Graph
=================================================

1. Add Edge
2. Remove Edge
3. Degree
4. Print Graph
5. Cek Edge
6. Keluar

=================================================
Masukkan Pilihan    : 1
=================================================

Masukkan asal       : 2
Masukkan tujuan     : 3
Masukkan jarak      : 30
```

2) Tambahkan method updateJarak pada Percobaan 1 yang digunakan untuk mengubah jarak antara dua node asal dan tujuan!

Pada class DoubleLinkedList_18

```
public void UpdateJarak(int data, int jarakBaru) {
    Node_18 current = head;
    while (current != null) {
        if (current.data == data) {
            current.jarak = jarakBaru;
            return;
        }
        current = current.next;
    }
    System.out.println("Tidak ada node dengan data " + data);
}
```

Pada class Graph_18

```
public void UpdateJarak(int asal, int tujuan, int jarakBaru) {
    list[asal].UpdateJarak(tujuan, jarakBaru);
}
```

3) Tambahkan method hitungEdge untuk menghitung banyaknya edge yang terdapat di dalam graf!

Pada class Graph_18

```
public int hitungEdge() {

    int totalEdge = 0;

    for (int i = 0; i < vertex; i++) {
        totalEdge += list[i].Size();
    }

    return totalEdge;

}
```

Repository : https://github.com/zidnafaz/Praktikum-Algoritma-Struktur-Data