**LAPORAN PRAKTIKUM**

**Politeknik Negeri Malang**

**Praktikan**

**ALGORITMA DAN STRUKTUR DATA**

****

2022

2141720183

RIDWAN CAESAR RIZQI KARISMA BIWARNI

TI 1C

Daftar Isi

[Percobaan Implementasi Graph menggunakan Linked List 4](#_Toc105660443)

[Langkah 1 4](#_Toc105660444)

[Langkah 2 4](#_Toc105660445)

[Langkah 3 4](#_Toc105660446)

[Langkah 4 5](#_Toc105660447)

[Langkah 5 5](#_Toc105660448)

[Langkah 6 5](#_Toc105660449)

[Langkah 7 6](#_Toc105660450)

[Langkah 8 6](#_Toc105660451)

[Langkah 9 6](#_Toc105660452)

[Langkah 10 7](#_Toc105660453)

[Langkah 11 7](#_Toc105660454)

[Langkah 12 8](#_Toc105660455)

[Langkah 13 8](#_Toc105660456)

[Langkah 14 9](#_Toc105660457)

[Pertanyaan Percobaan 10](#_Toc105660458)

[Jawaban Percobaan 10](#_Toc105660459)

[Percobaan Implementasi Graph menggunakan Matriks 12](#_Toc105660460)

[Langkah 1 12](#_Toc105660461)

[Langkah 2 12](#_Toc105660462)

[Langkah 3 12](#_Toc105660463)

[Langkah 4 13](#_Toc105660464)

[Langkah 5 13](#_Toc105660465)

[Verifikasi Hasil Percobaan 14](#_Toc105660466)

[Pertanyaan Percobaan 14](#_Toc105660467)

[Jawaban Percobaan 14](#_Toc105660468)

[Tugas Praktikum 16](#_Toc105660469)

[Soal Nomor 1 16](#_Toc105660470)

[Jawaban 16](#_Toc105660471)

[Source code 16](#_Toc105660472)

[Output 16](#_Toc105660473)

[Soal Nomor 2 17](#_Toc105660474)

[Jawaban 17](#_Toc105660475)

[Source code yang dimodifikasi 17](#_Toc105660476)

[Output Undirected 19](#_Toc105660477)

[Output Directed 19](#_Toc105660478)

[Soal Nomor 3 19](#_Toc105660479)

[Jawaban 20](#_Toc105660480)

[Source code method remove setelah dimodifikasi 20](#_Toc105660481)

[Source code method removeEdge setelah direvisi 20](#_Toc105660482)

[Output 21](#_Toc105660483)

[Soal Nomor 4 21](#_Toc105660484)

[Jawaban 21](#_Toc105660485)

[Source code graph 21](#_Toc105660486)

[Source code graphArray 23](#_Toc105660487)

[Output graph 25](#_Toc105660488)

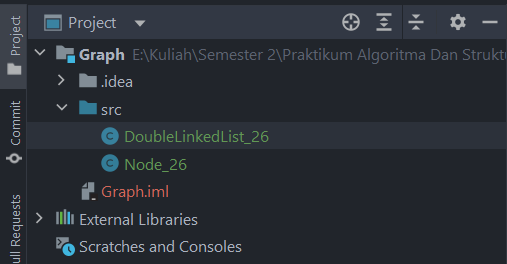
[Output graphArray 25](#_Toc105660489)

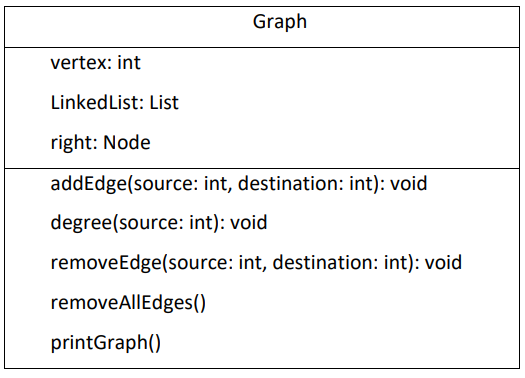
# Percobaan Implementasi Graph menggunakan Linked List

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

## Langkah 1

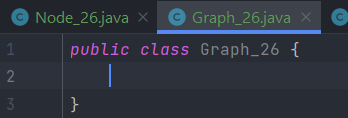
Buatlah class Node, dan class Linked Lists sesuai dengan praktikum Double Linked Lists.





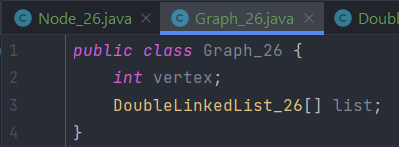
## Langkah 2

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



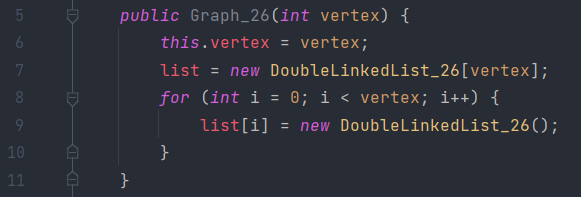
## Langkah 3

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



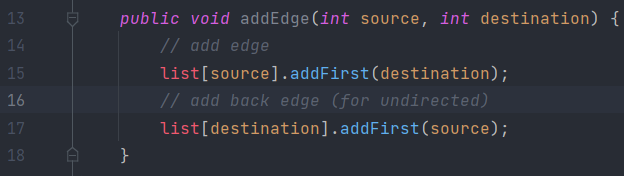
## Langkah 4

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



## Langkah 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().



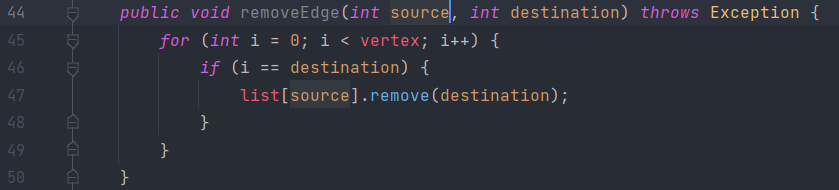
## Langkah 6

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



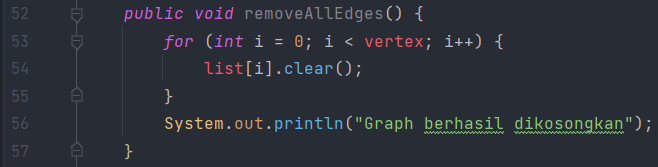
## Langkah 7

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



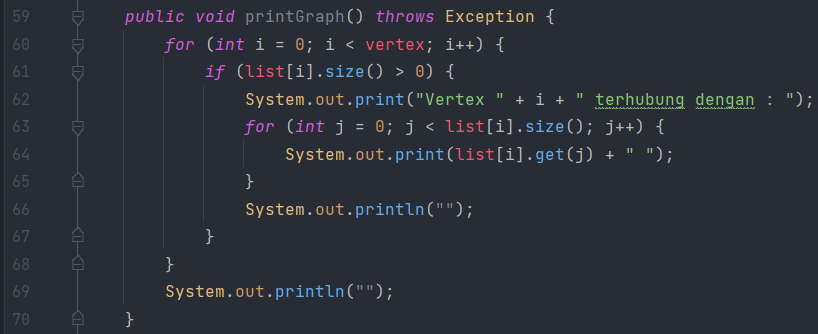
## Langkah 8

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



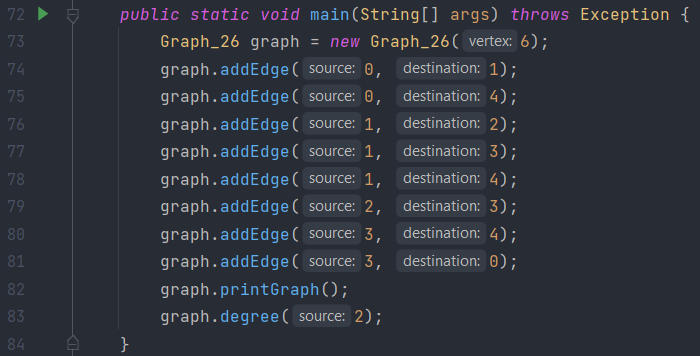
## Langkah 9

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



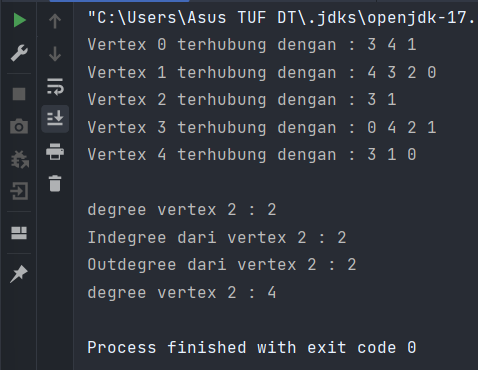
## Langkah 10

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).



## Langkah 11

Amati hasil running tersebut



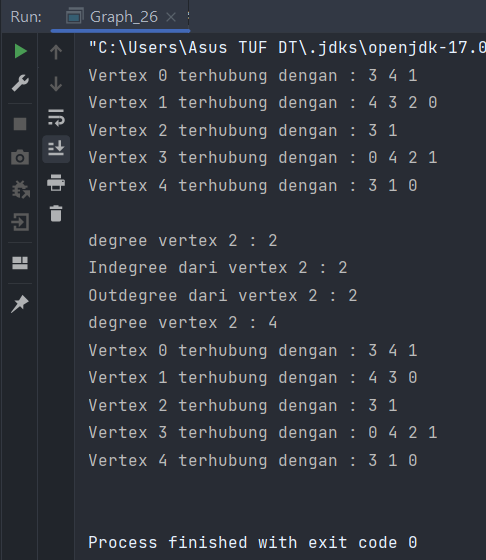
## Langkah 12

Tambahkan pemanggilan method removeEdge() sesuai potongan code di bawah ini pada method main(). Kemudian tampilkan graph tersebut.



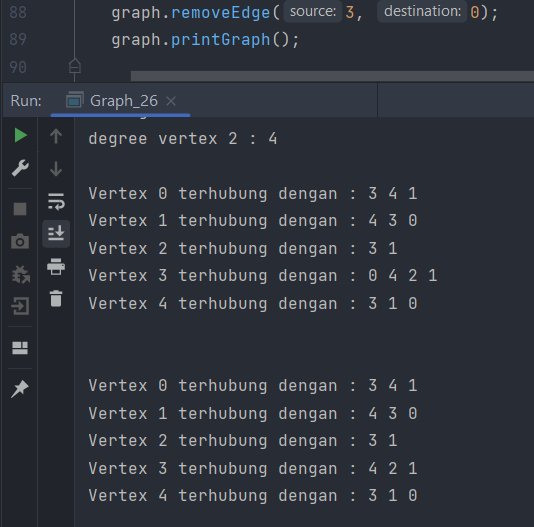
## Langkah 13

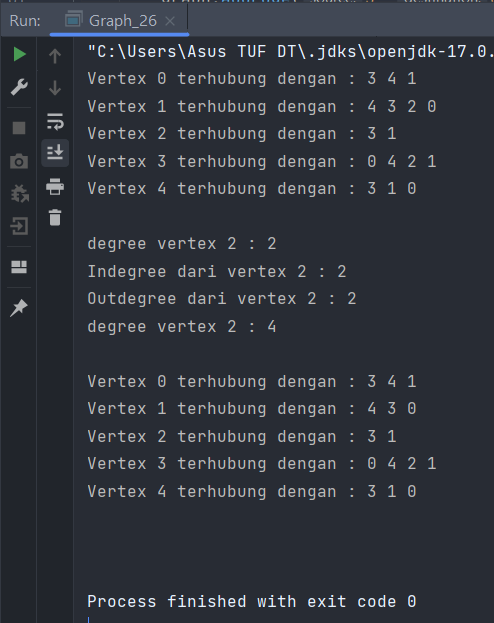
Amati hasil running tersebut.



## Langkah 14

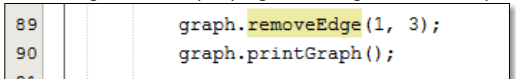
Uji coba penghapusan lintasan yang lain! Amati hasilnya!



Verifikasi Hasil Percobaan

### 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 ?



### Jawaban Percobaan

1. Algoritma yang menggunakan dasar graph.
   1. Algoritma Prim

Kegunaan: untuk mencari pohon rentang minimum untuk sebuah graf berbobot yang saling terhubung.

* 1. Algoritma KNN atau K-Nearest Neighbor

Kegunaan: mengklasifikasikan data berdasarkan similarity atau kemiripan atau kedekatannya terhadap data lainnya.

* 1. Algoritma Welch Powell

Kegunaan: algoritma yang digunakan untuk mewarnai sebuah graf secara mangkus, dan tidak selalu memberikan warna minimum untuk sebuah graf, namun keuntungan dari algoritma ini ialah penggunaannya yang lebih sederhana dan lebih mudah.

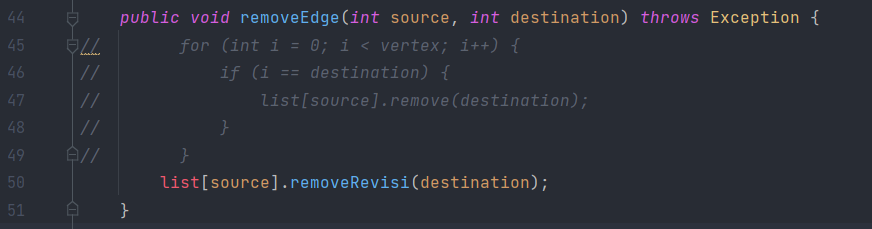
* 1. Algoritme Dijkstra

Kegunaan: Sebuah algoritma yang dipakai dalam memecahkan permasalahan jarak terpendek (shortest path problem) untuk sebuah graf berarah (directed graph).

1. Variabel array bertipe linked list digunakan untuk menyimpan setiap node vertex pada masing masing index array linkedlist.
2. Karena agar vertex yang dihubungkan oleh edge berada urut didepan vertex source.
3. Pada kasus ini menggunakan pemberian index pada method remove pada class doublelinkedlist dengan parameter destination sebagai acuan.
4. Karena parameter destination ketika pemanggilan method remove dari class doublelinkedlist akan digunakan sebagai index. Sehingga yang dihapus adalah sesuai urutan index tidak sesuai dengan value vertex sehingga muncul error indeks diluar batas ketika ingin menghapus edge dengan source 1 dan destination 3.

Solusi:

Disini saya sedikit memodifikasi method remove yang semula menggunakan indeks disini saya menggunakan destination parameter sebagai acuan node mana yang akan dihapus sehingga tidak ada indeks melebihi batas.



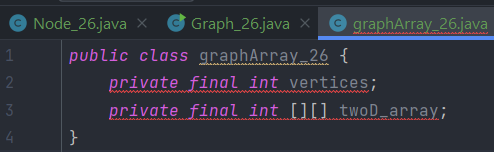


# Percobaan Implementasi Graph menggunakan Matriks

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

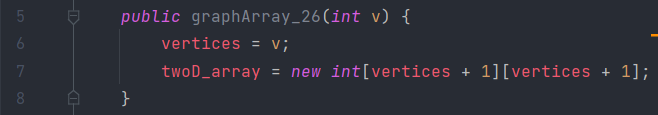
## Langkah 1

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!



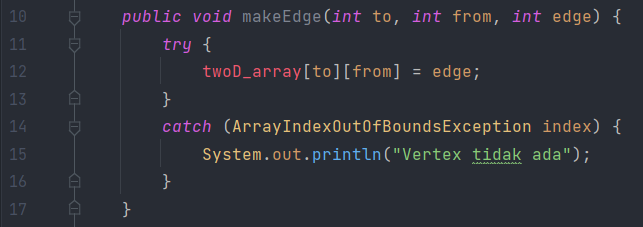
## Langkah 2

Buatlah konstruktor graphArray sebagai berikut!

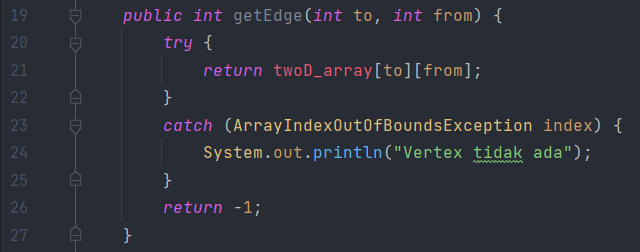


## Langkah 3

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



Untuk menampilkan suatu lintasan diperlukan pembuatan method getEdge() berikut



## Langkah 4

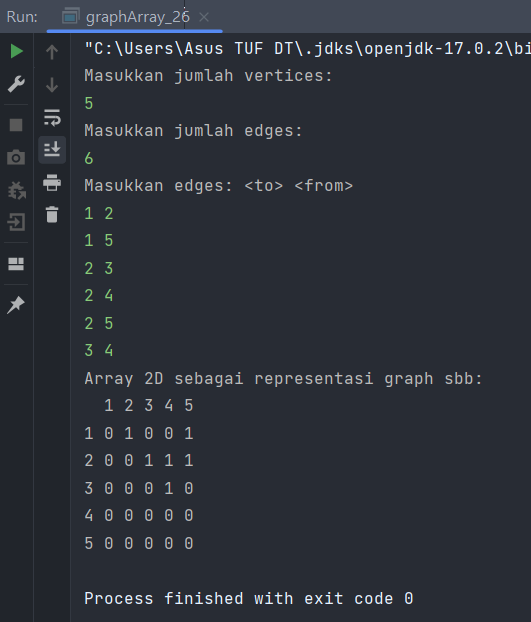
Kemudian buatlah method main() seperti berikut ini.

*public static void* main(String[] args) {  
 *int* v, e, count = 1, to = 0, from = 0;  
 Scanner sc = *new* Scanner(System.in);  
 graphArray\_26 graph;  
 *try* {  
 System.out.println("Masukkan jumlah vertices: ");  
 v = sc.nextInt();  
 System.out.println("Masukkan jumlah edges: ");  
 e = sc.nextInt();  
  
 graph = *new* graphArray\_26(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();  
}

## Langkah 5

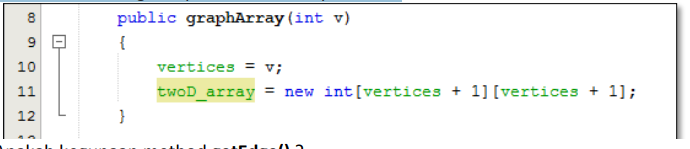
Jalankan class graphArray dan amati hasilnya!

## Verifikasi Hasil Percobaan



### 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?



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

### Jawaban Percobaan

1. Degree pada undirected graph jika ada edge menghubungkan antar vertex maka dihitung 1 degree sedangkan degree pada directed graph edge akan dihitung 1 jika menghubungkan antar vertex tidak secara bolak-balik, jika edge menghubungkan vertex A dengan vertex B secara bolak-balik maka akan dihitung 2 degree.
2. Karena vertex dimulai dari angka 1 jadi array harus ditambah 1.
3. Untuk menampilkan nilai array pada indeks ke[to][from] namun jika indeks out of bound maka akan muncul pemberitahuan vertex tidak ada.
4. Directed graph.
5. Agar ketika ada error program tidak langsung close namun ada pemberitahuan untuk mengecek errornya.

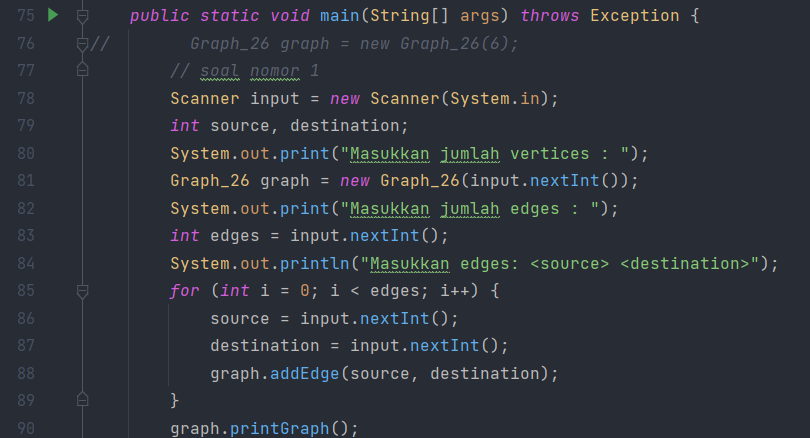
# Tugas Praktikum

## Soal Nomor 1

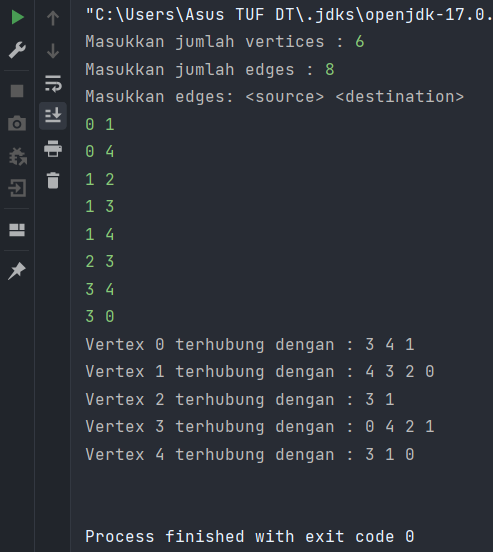
Ubahlah lintasan pada praktikum 2.1 menjadi inputan!

## Jawaban

### Source code



### Output



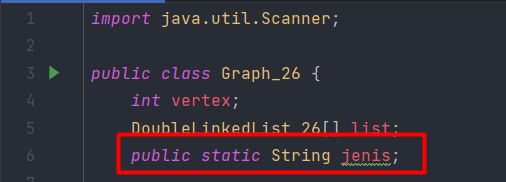
## Soal Nomor 2

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

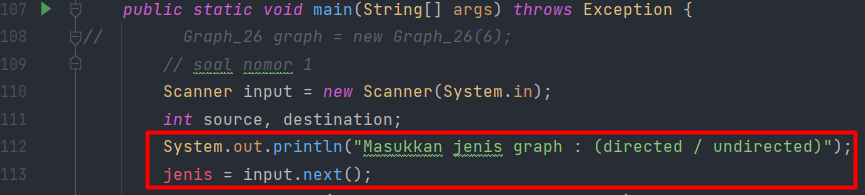
## Jawaban

### Source code yang dimodifikasi

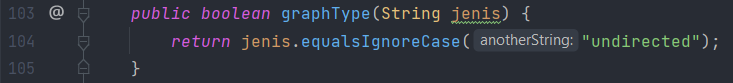
Penambahan variabel jenis



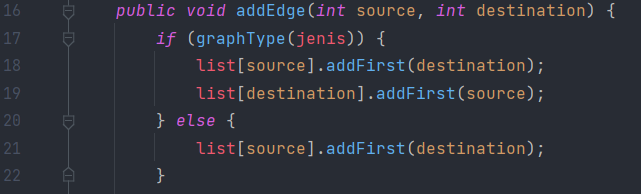
Input value variabel jenis



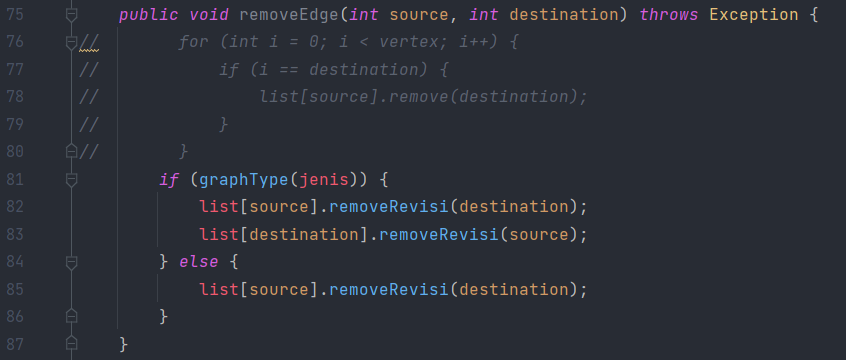
Method mengecek jenis graph



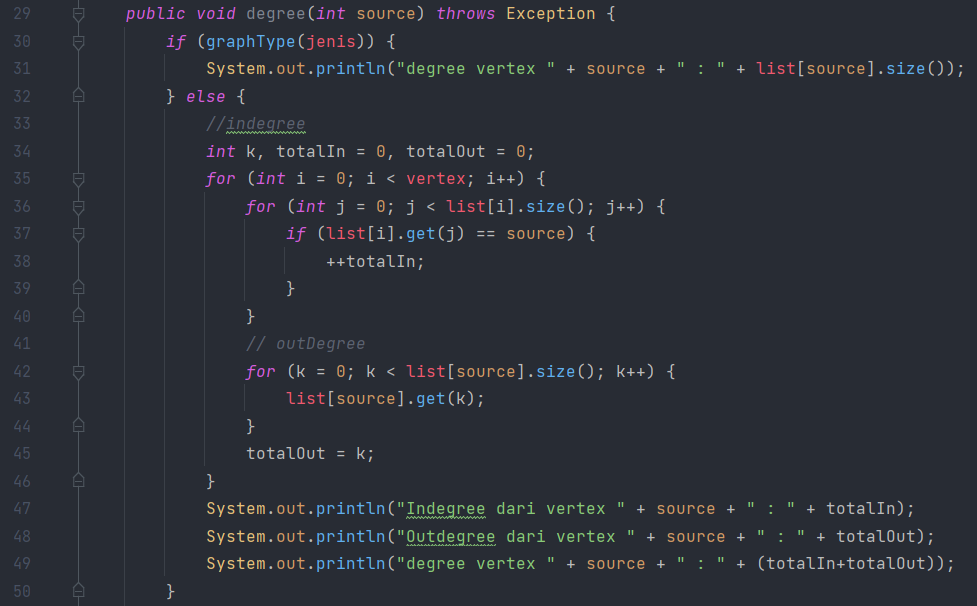
Modifikasi pada addEdge



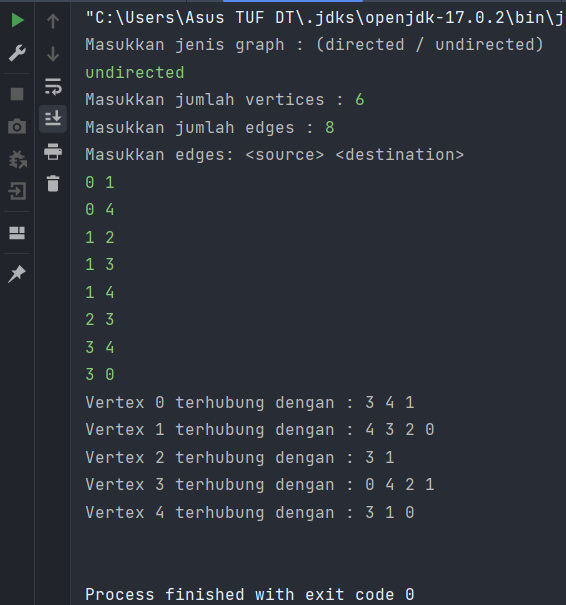
Modifikasi pada removeEdge



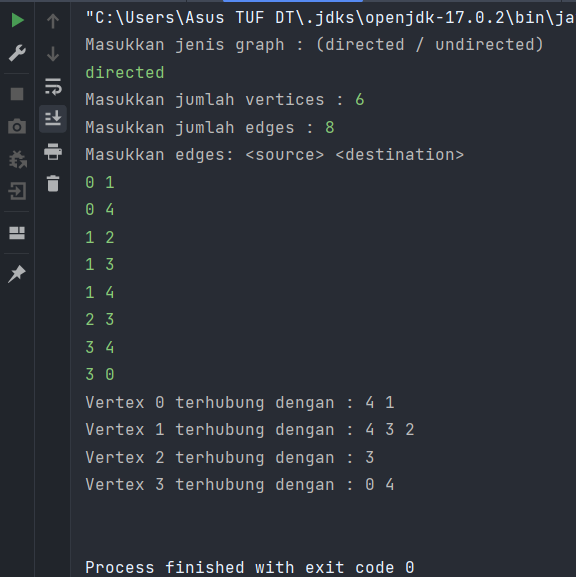
Modifikasi pada degree



### Output Undirected



### Output Directed



## Soal Nomor 3

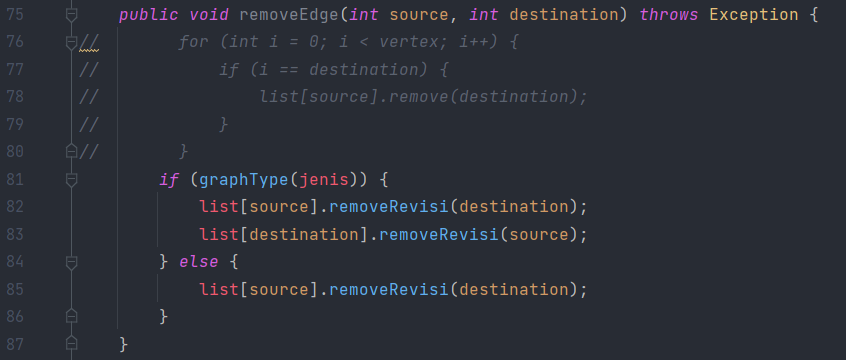
Modifikasi method removeEdge() pada praktikum 2.1 agar tidak menghasilkan output yang salah untuk path selain path pertama kali!

## Jawaban

### Source code method remove setelah dimodifikasi



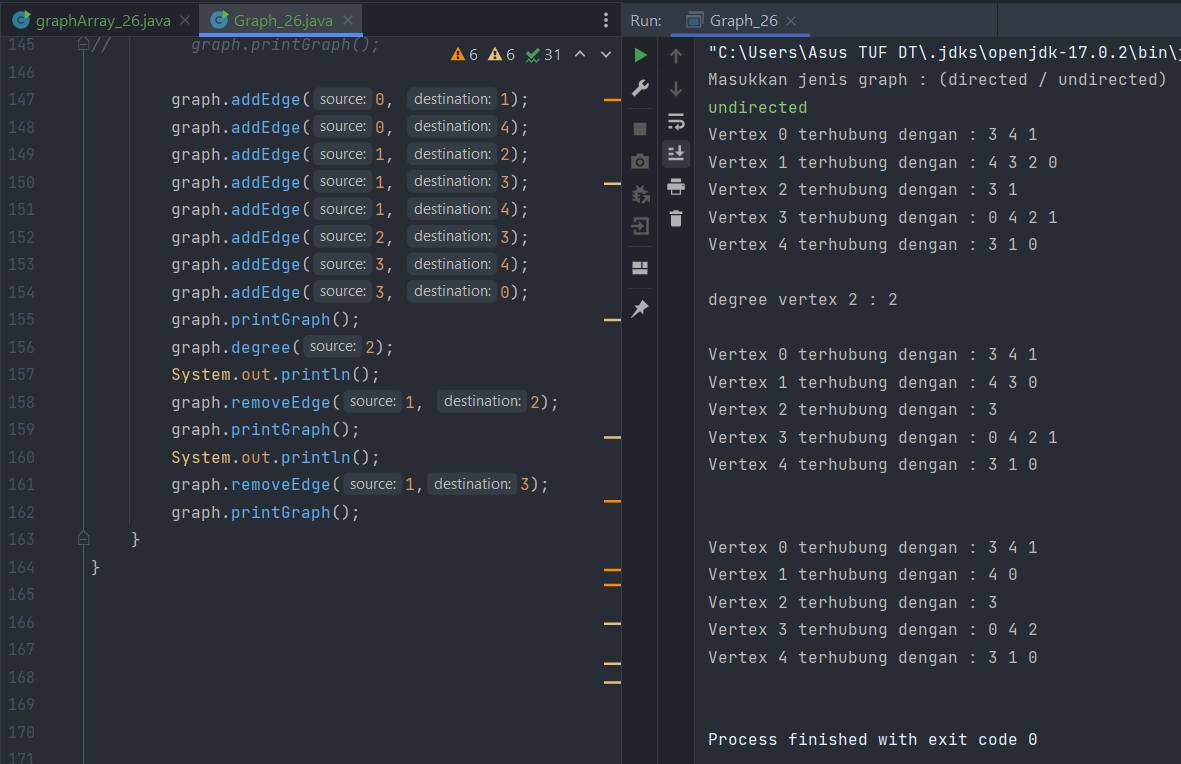
### Source code method removeEdge setelah direvisi



### Output

Dengan inputan yang sama seperti praktikum 2.1 namun sudah bisa menghapus edge selain pertama kali.

Disini sudah terlihat bahwa pada vertex 1 sudah tidak terhubung ke vertex 2 dan 3 dan sebaliknya karena menggunakan graph undirected.



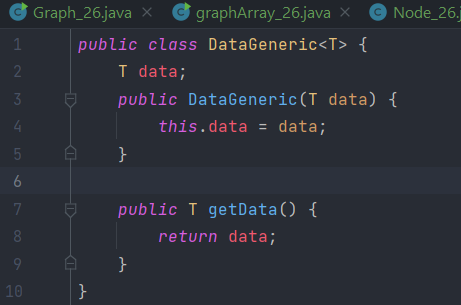
## Soal Nomor 4

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.

## Jawaban

### Source code graph

Membuat class generic



Modifikasi method main



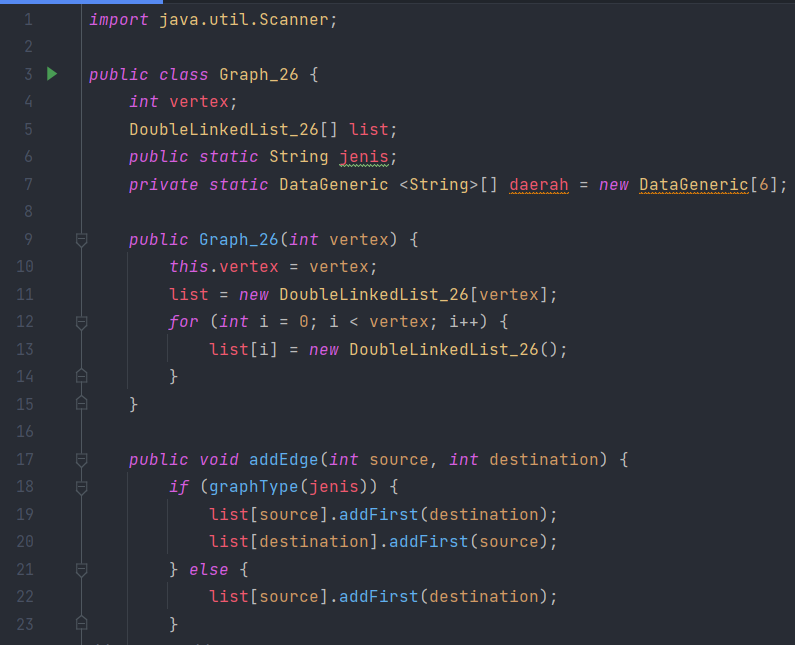
Modifikasi method printGraph



Modifikasi method degree

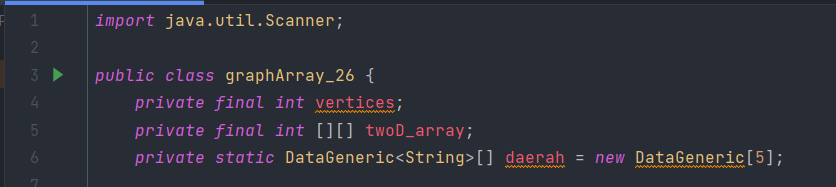


Menambahkan variabel generic dan modifikasi class addEdge



### Source code graphArray

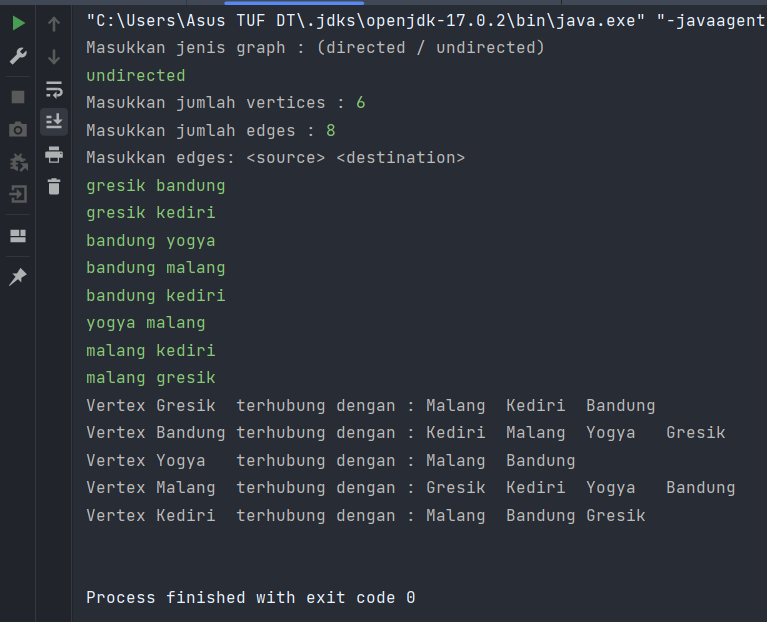
Menambah variabel generic



Modifikasi method main

*public static void* main(String[] args) {  
 daerah[0] = *new* DataGeneric<>("Gresik ");  
 daerah[1] = *new* DataGeneric<>("Bandung");  
 daerah[2] = *new* DataGeneric<>("Yogya ");  
 daerah[3] = *new* DataGeneric<>("Malang ");  
 daerah[4] = *new* DataGeneric<>("Kediri ");  
 *int* v, e, count = 1, indexTo = 0, indexFrom = 0;  
 String to, from;  
 Scanner sc = *new* Scanner(System.in);  
 graphArray\_26 graph;  
 *try* {  
 System.out.println("Masukkan jumlah vertices: ");  
 v = sc.nextInt();  
 System.out.println("Masukkan jumlah edges: ");  
 e = sc.nextInt();  
 graph = *new* graphArray\_26(v);  
 System.out.println("Masukkan edges: <to> <from>");  
 *while* (count <= e) {  
 to = sc.next();  
 from = sc.next();  
 *for* (*int* i = 0; i < v; i++) {  
 *if* (to.equalsIgnoreCase(daerah[i].getData().trim())) {  
 indexTo = i;  
 }  
 *if* (from.equalsIgnoreCase(daerah[i].getData().trim())) {  
 indexFrom = i;  
 }  
 }  
 graph.makeEdge(indexTo, indexFrom + 1, 1);  
 count++;  
 }  
 System.out.println("Array 2D sebagai representasi graph sbb: ");  
 System.out.print("\t\t ");  
 *for* (*int* i = 0; i < v; i++) {  
 System.out.print(daerah[i].getData() + " ");  
 }  
 System.out.println();  
  
 *for* (*int* i = 0; i < v; i++) {  
 System.out.print(daerah[i].getData() + " \t");  
 *for* (*int* j = 1; j <= v; j++) {  
 System.out.print(graph.getEdge(i, j) + " \t\t");  
 }  
 System.out.println();  
 }  
 } *catch* (Exception E) {  
 System.out.println("Error. Silakan cek kembali\n" +E.getMessage());  
 }  
 sc.close();  
}

### Output graph



### Output graphArray

