# JOBSHEET XII

**Graph**

# 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

# Praktikum

* 1. **Percobaan 1: Implementasi Graph menggunakan Linked List**

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 |

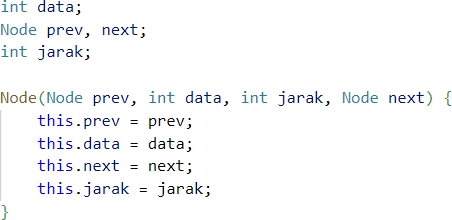
# Langkah-langkah Percobaan Waktu percobaan (90 menit)

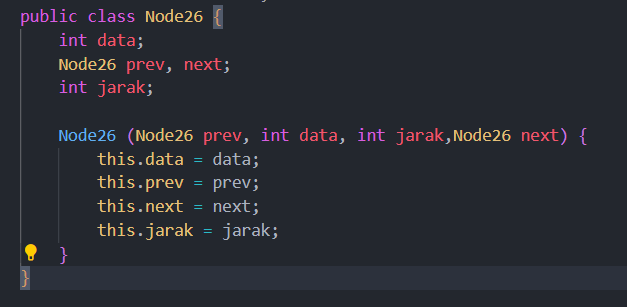
* + - 1. Buka text editor. Buat class **Node<NoAbsen>.java** dan class

**DoubleLinkedList<NoAbsen>.java** sesuai dengan praktikum **Double Linked List.**

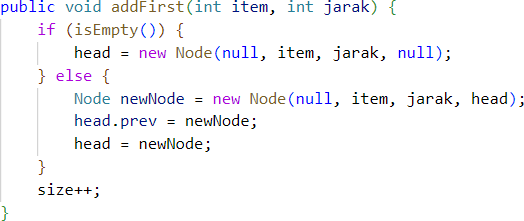
# Class Node

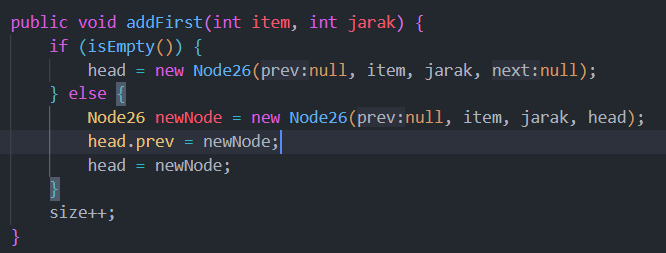
Kode program yang terdapat pada class **Node** belum dapat mengakomodasi kebutuhan pembuatan graf berbobot, sehingga diperlukan sedikit modifikasi. Setelah Anda menyalin kode program dari class **Node** pada praktikum Double Linked List, tambahkan atribut **jarak** bertipe **int** untuk menyimpan bobot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





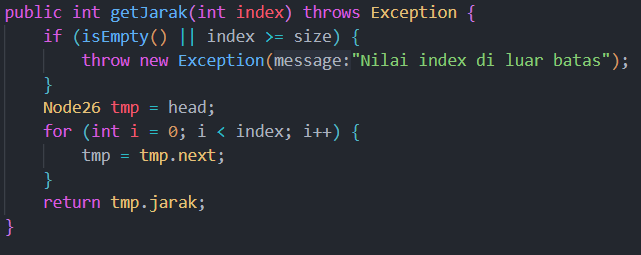
# Class DoubleLinkedList

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

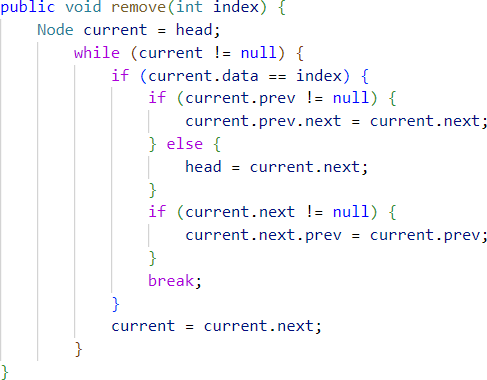


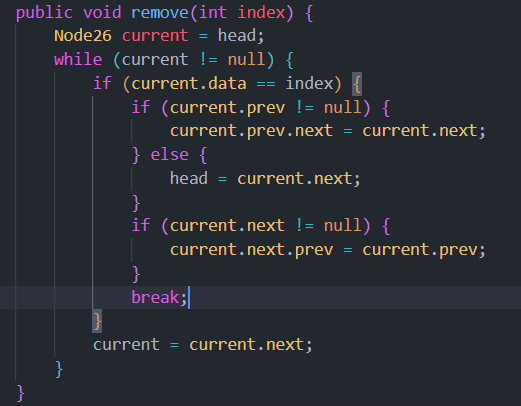
Selanjutnya buat method **getJarak** (hampir sama seperti method **get**) yang digunakan untuk mendapatkan nilai jarak edge antara dua node.





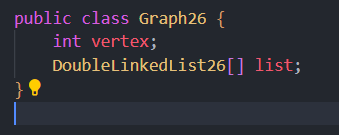
Modifikasi method **remove** agar dapat melakukan penghapusan edge sesuai dengan node asal dan tujuan pada graf. Pada praktikum Double Linked List, parameter **index** digunakan untuk menghapus data sesuai posisi pada indeks tertentu, sedangkan pada Graf ini, penghapusan didasarkan pada data node tujuan,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



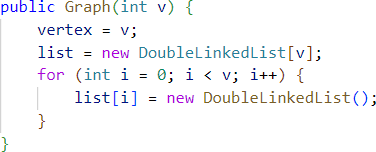


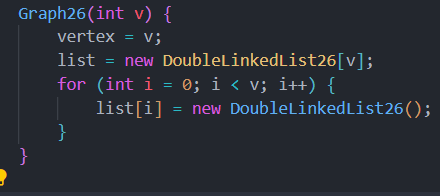
# Class Graph

* + - 1. Buat file baru, beri nama **Graph<NoAbsen>.java**
      2. Lengkapi class **Graph** dengan atribut yang telah digambarkan di dalam pada class diagram, yang terdiri dari atribut **vertex** dan **DoubleLinkedList**

****

* + - 1. Tambahkan konstruktor default untuk menginisialisasi variabel **vertex** dan menambahkan perulangan jumlah vertex sesuai dengan panjang array yang telah ditentukan.

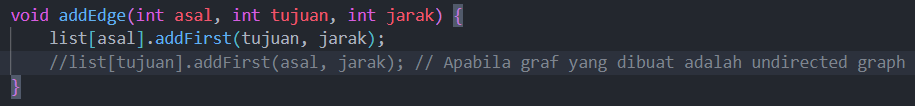




* + - 1. Tambahkan method **addEdge()** untuk menghubungkan dua node. Baris kode program berikut digunakan untuk graf berarah (directed).

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



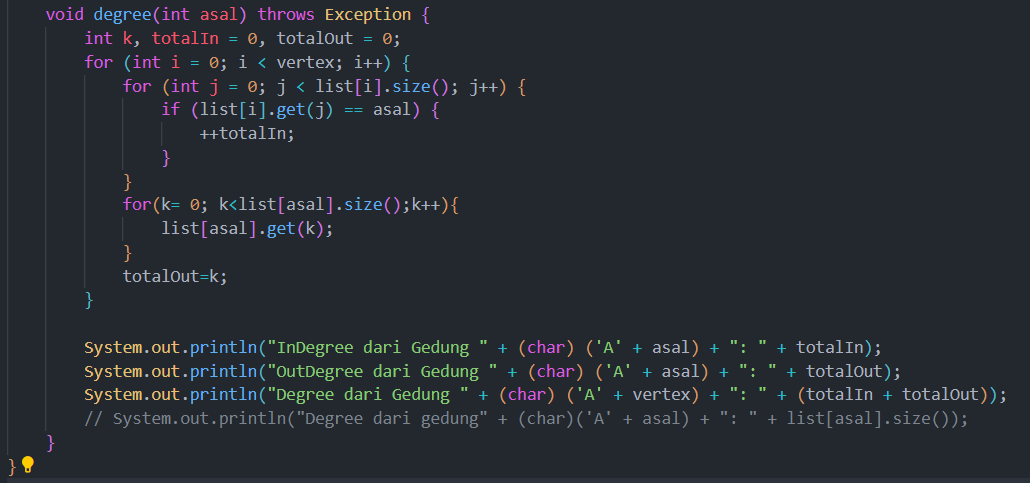


* + - 1. Tambahkan method **degree()** untuk menampilkan jumlah derajat lintasan pada setiap vertex. Kode program berikut digunakan untuk menghitung degree pada graf berarah

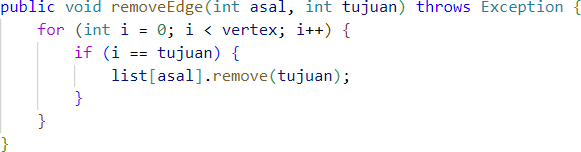


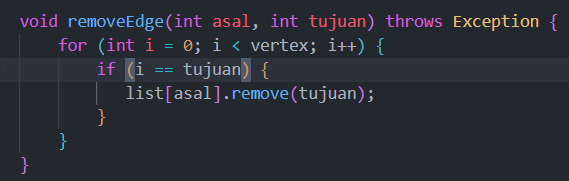
Apabila graf yang dibuat adalah undirected graph, maka cukup gunakan kode berikut.

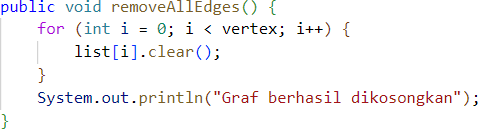


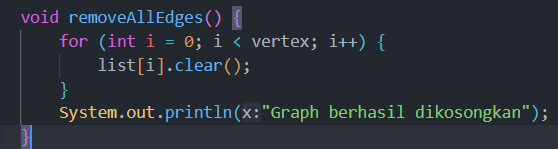


* + - 1. Tambahkan method **removeEdge()** untuk menghapus lintasan pada suatu graph. Penghapusan membutuhkan 2 parameter yaitu node **asal** dan **tujuan**.

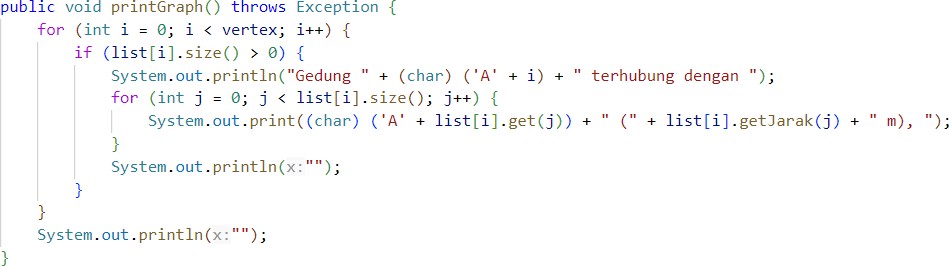


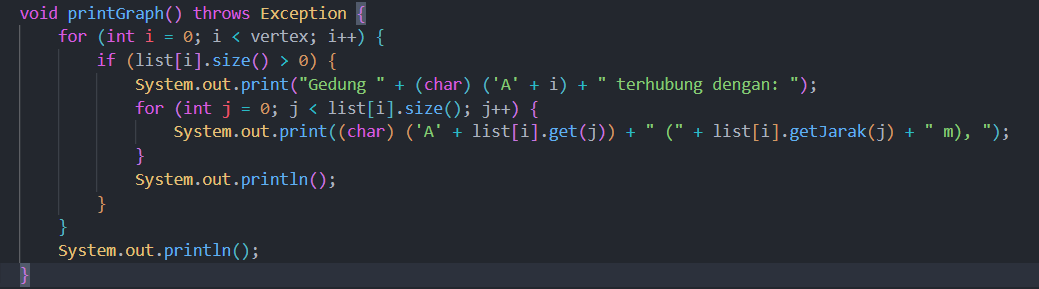


* + - 1. Tambahkan method **removeAllEdges()** untuk menghapus semua vertex yang ada di dalam graf.



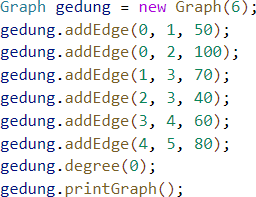
* + - 1. Tambahkan method **printGraph()** untuk mencetak graf.

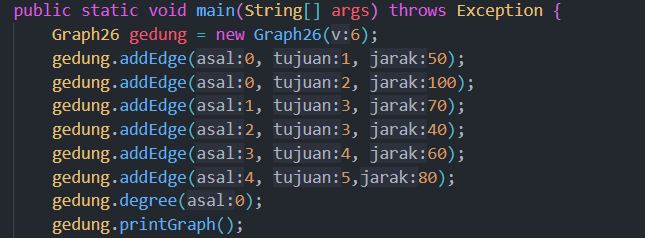




# D. Class Utama

* + - 1. Buat file baru, beri nama **GraphMain<NoAbsen>.java**
      2. Tuliskan struktur dasar bahasa pemrograman Java yang terdiri dari fungsi **main**
      3. Di dalam fungsi **main**, lakukan instansiasi object Graph bernama **gedung** dengan nilai parameternya adalah 6.
      4. Tambahkan beberapa edge pada graf, tampilkan degree salah satu node, kemudian tampilkan grafnya.





* + - 1. Compile dan run program.

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

* + - 1. Tambahkan pemanggilan method **removeEdge()**, kemudian tampilkan kembali graf tersebut.



# Commit dan push kode program ke Github

* + - 1. Compile dan run program.

# Verifikasi Hasil Percobaan

Verifikasi hasil kompilasi kode program Anda dengan gambar berikut ini.

|  |  |
| --- | --- |
| **Hasil running pada langkah 14** | **Hasil running pada langkah 17** |
| A screenshot of a computer  Description automatically generated |  |
|  |  |
|  |  |

# Pertanyaan

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!

Atribut list[] digunakan untuk menyimpan daftar tetangga (adjacency list) dari setiap node dalam graf. Dengan menggunakan DoubleLinkedList, operasi penambahan dan penghapusan edge menjadi lebih efisien.

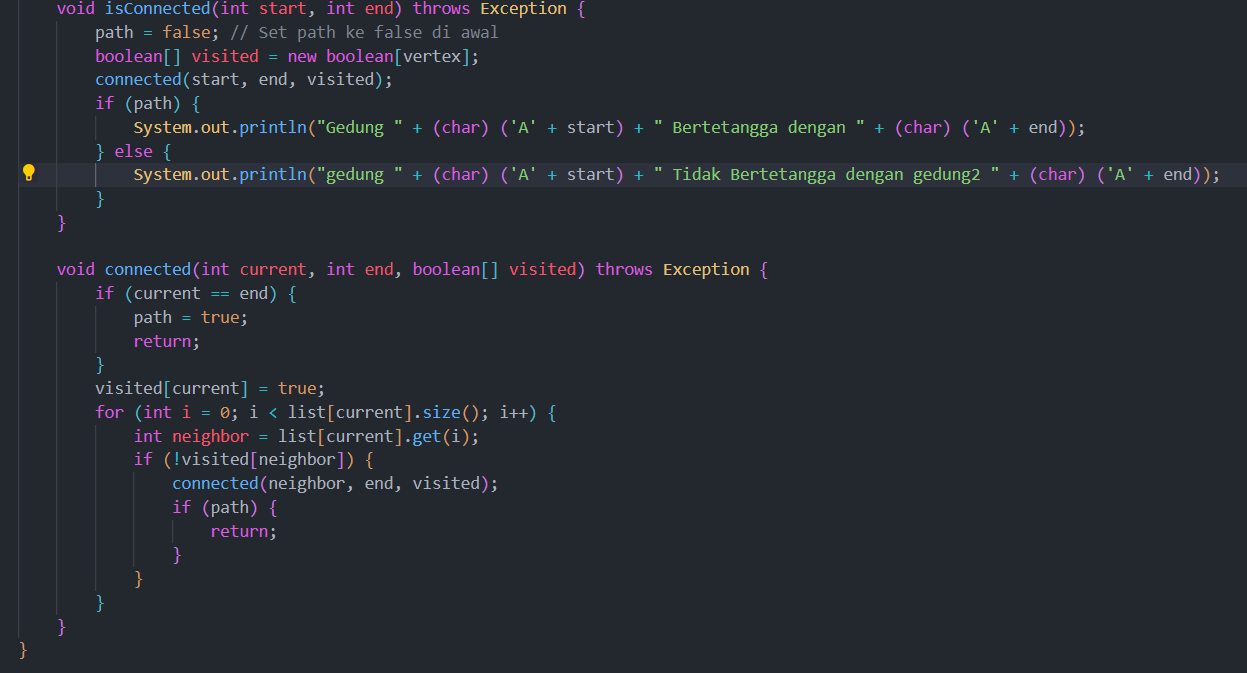
1. Jelaskan alur kerja dari method **removeEdge**!

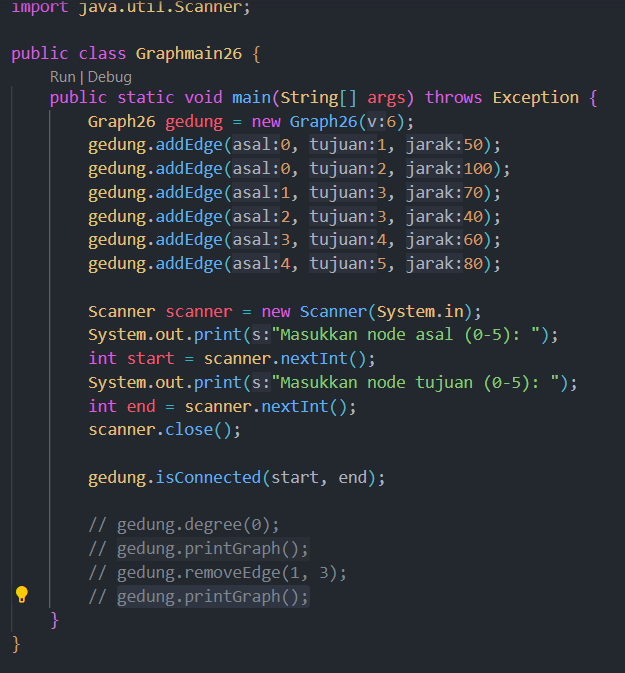
* Menentukan node asal dan node tujuan dari edge yang akan dihapus.
* Mengakses adjacency list dari node asal.
* Mencari dan menghapus node tujuan dalam adjacency list tersebut.
* Jika graf tidak berarah, ulangi langkah yang sama untuk adjacency list dari node tujuan.

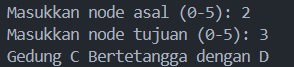
1. Apakah alasan pemanggilan method **addFirst()** untuk menambahkan data, bukan method add jenis lain saat digunakan pada method **addEdge** pada class Graph?

addFirst() digunakan untuk menambahkan node baru di awal DoubleLinkedList agar operasi penambahan lebih cepat (O(1) waktu). Metode ini lebih efisien dibandingkan metode penambahan lain yang mungkin memerlukan traversal seluruh daftar (O(n) waktu).

1. A close up of black text  Description automatically generatedA close up of black text  Description automatically generatedModifikasi kode program sehingga dapat dilakukan pengecekan apakah terdapat jalur antara suatu node dengan node lainnya, seperti contoh berikut (Anda dapat memanfaatkan Scanner).





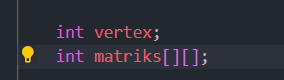
# Percobaan 2: Implementasi Graph menggunakan Matriks

Dengan menggunakan kasus yang sama dengan Percobaan 1, pada percobaan ini implementasi graf dilakukan dengan menggunakan matriks dua dimensi.

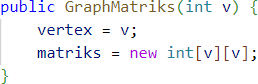
# Langkah-langkah Percobaan Waktu percobaan: 60 menit

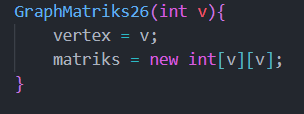
1. Buat file baru, beri nama **GraphMatriks<NoAbsen>.java**
2. Lengkapi class **GraphMatriks** dengan atribut **vertex** dan **matriks**





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





1. Untuk membuat suatu lintasan yang menghubungkan dua node, maka dibuat method

**makeEdge()** sebagai berikut.

A black and blue text  Description automatically generated

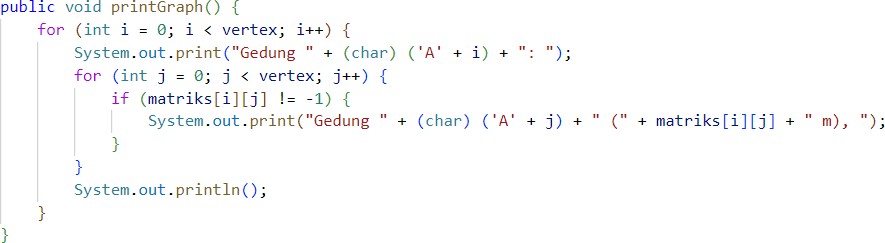


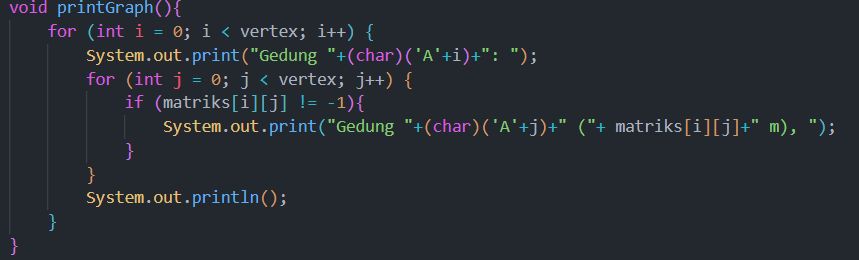
1. Tambahkan method **removeEdge()** untuk menghapus lintasan pada suatu graf.

A white background with black and blue text  Description automatically generated

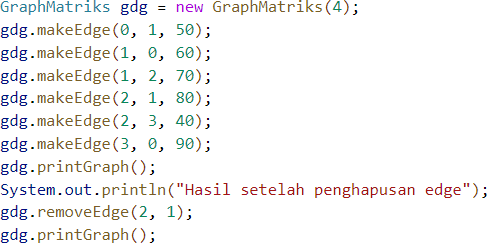


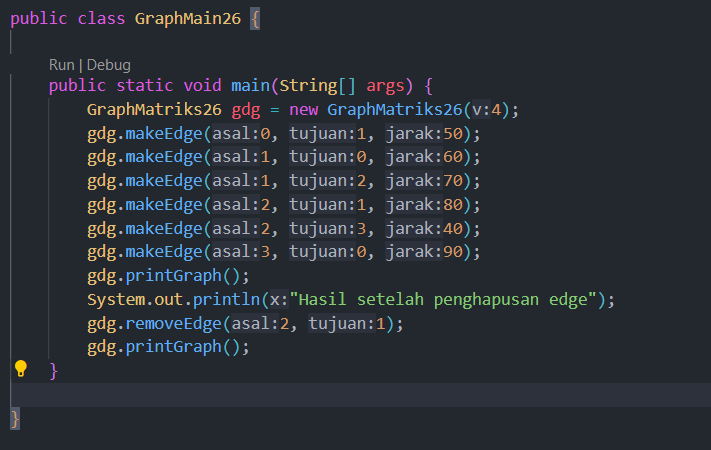
1. Tambahkan method printGraph() untuk mencetak graf.





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



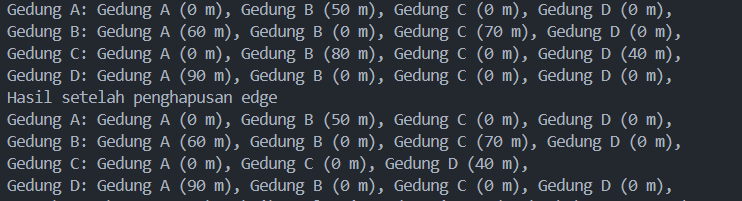
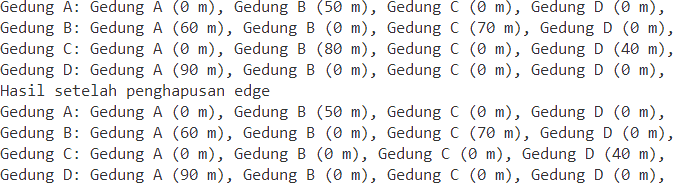


# Commit dan push kode program ke Github

1. Compile dan run program.

# Verifikasi Hasil Percobaan

Verifikasi hasil kompilasi kode program Anda dengan gambar berikut ini.

# Pertanyaan

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

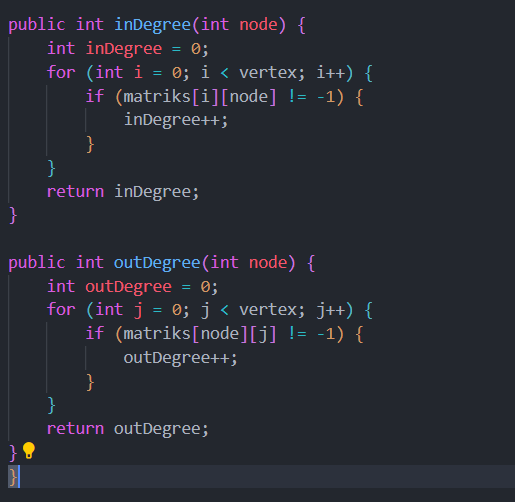
Graph yang digunakan adalah directed weighted graph (graf terarah berbobot).

* + - 1. Apa maksud dari dua baris kode berikut?

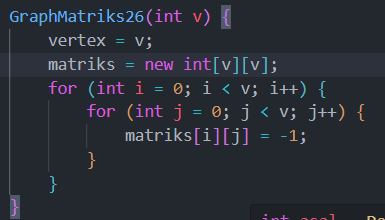


Baris pertama (gdg.makeEdge(1, 2, 70)) menambahkan edge dari node 1 ke node 2 dengan bobot 70. Baris kedua (gdg.makeEdge(2, 1, 80)) menambahkan edge dari node 2 ke node 1 dengan bobot 80. Ini menunjukkan bahwa graf adalah terarah dan bobot antara dua node

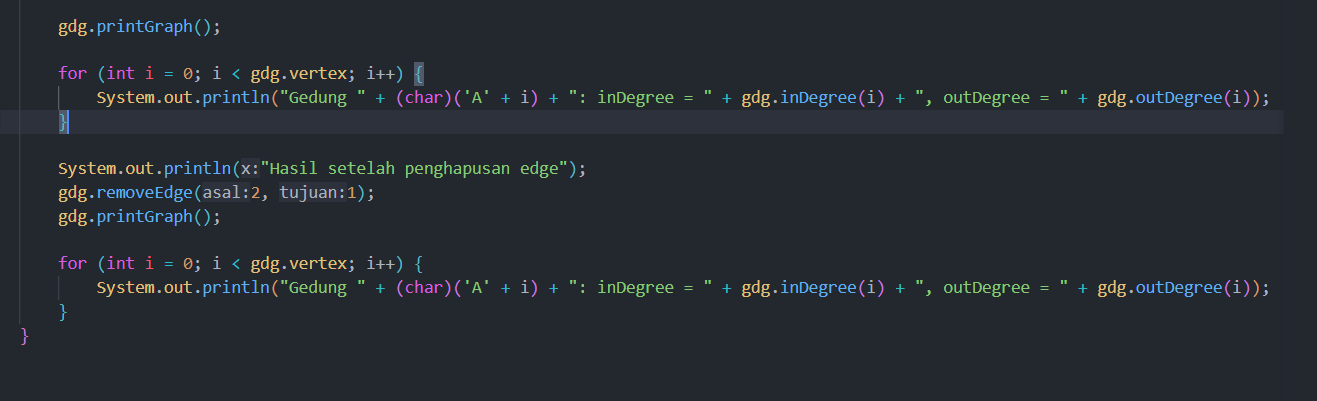
* + - 1. Modifikasi kode program sehingga terdapat method untuk menghitung degree, termasuk inDegree dan outDegree!

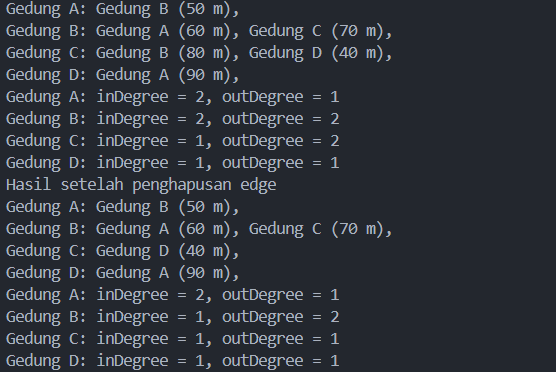


GraphMatriks26:



GraphMain26:

  
Hasil:



# 3. Latihan Praktikum

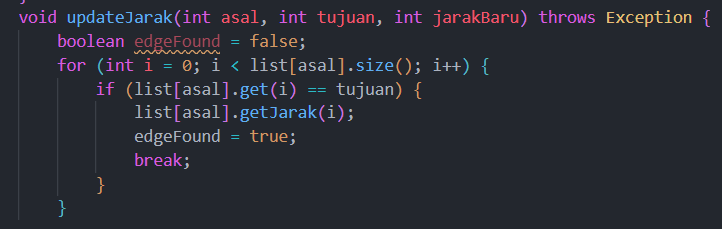
**Waktu percobaan: 90 menit**

1. Modifikasi kode program pada class **GraphMain** sehingga terdapat menu program yang bersifat dinamis, setidaknya terdiri dari:
   1. Add Edge
   2. Remove Edge
   3. Degree
   4. Print Graph
   5. Cek Edge

Pengguna dapat memilih menu program melalui input Scanner



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



1. Tambahkan method **hitungEdge** untuk menghitung banyaknya edge yang terdapat di dalam graf!

