Name : Azaria Cindy Sahasika
Number Id : 2341760169 / 06
Class : 1G – Business Information System
Lesson : Algorithm and Data Structure
Material : Material 12 – Graph
Github Link : https://github.com/azariacindy/algorithm-ds

JOBSHEET XII
Graph

12.1 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.

12.2 Praktikum

12.1 Implementasi Graph menggunakan Linked List

12.1.1 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!

```
Vertex 0 connected with: 3 4 1
Vertex 1 connected with: 4 3 2 0
Vertex 2 connected with: 3 1
Vertex 3 connected with: 0 4 2 1
Vertex 4 connected with: 3 1 0

degree vertex 2 : 2
Indegree from vertex 2 : 2
Outdegree from vertex 2 : 2
degree vertex 2 : 4

Vertex 0 connected with: 3 4 1
Vertex 1 connected with: 4 3 0
Vertex 2 connected with: 3 1
Vertex 3 connected with: 0 4 2 1
Vertex 4 connected with: 3 1 0
```

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?
   - Breadth-First Search (BFS), untuk traversing atau mencari semua node di sebuah graph secara level-wise. Biasa digunakan untuk menemukan jarak terpendek pada graph tak berbobot dan dalam melakukan pencarian pada jaringan yang sangat luas seperti web crawling.
   - Depth-First Search (DFS), untuk traversing atau mencari semua node di sebuah graph secara depth-wise. Biasa digunakan untuk algoritma penyelesaian masalah seperti menemukan komponen terhubung, topologi sorting dan menemukan semua path dalam suatu graph.
   - Dijkstra's Algorithm, untuk menemukan jalur terpendek dari satu node ke node lainnya dalam graph berbobot yang tidak memiliki bobot tepi negatif. Sngat berguna dalam aplikasi seperi routing jaringan dan sistem navigasi.

2. Pada class Graph terdapat array bertipe LinkedList, yaitu LinkedList list[]. Apakah tujuan pembuatan variabel tersebut?
   ➔ Bertujuan untuk merepresentasikan adjacency list dari sebuah graph. Setiap elemen array adalah linked list yang menyimpan semua node tetangga dari node tertentu. Hal ini merupakan cara yang efisien untuk menyimpan dan mengelola struktur data graph karena memungkinkan penambahan dan penghapusan edge yang cepat dan traversal yang efektif.

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?**
   ➔ **'addFirst()'** digunakan untuk menambahkan elemen baru diawal linked list, juga digunakan untuk memastikan bahwa penambahan edge yang dilakukan dengan cara yang efisien, karena penambahan diawal linked list memiliki kompleksitias O(1). Hal ini juga memastikan jika urutan penambahan edge adalah konsisten dan predictable.

4. Bagaimana cara mendeteksi prev pointer pada saat akan melakukan penghapusan suatu edge pada graph ?
   ➔ Untuk mendeteksi prev pointer pada saat penghapusan edge dalam adjacency list yang direpresentasikan dengan linked list, kita perlu traversal linked list tersebut dari head hingga menemukan node yang akan dihapus. Selama traversal, kita simpan referensi ke node sebelumnya (prev). Setelah menemukan node yang akan dihapus, kita atur prev.next ke current.next untuk menghapus node dari linked list.

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();
```

   ➔ Mungkin karena tidak eksis atau penghapusan edge dilakukan pada adjacency list yang tidak singkron dengan graph sebenarnya. Bisa menyebablan inkonsistensi pada representasi graph.

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.

```
Enter number of vertices:
5
Enter number of edges:
6
Insert edges: <to> <from>
1 2
1 5
2 3
2 4
2 5
3 4
2D array as graph representation as follows:
  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
```

```
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*?
   - Undirected graph, degree adalah jumlah edge yang terhubung ke sebuah node.
   - Directed grph
     a. In-degree, jumlah edge yang masuk ke sebuah node
     b. Out-degree, jumlah edge yang keluar dari sebuah node

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      }
```

   - Untuk membuat adjecency matrix yang indeksnya mulai dari 1, bukan dari 0. Hal ini untuk memudahkan pemahaman dan manipulasi graph berdasarkan indeks yang lebih intuitif. Misal, jika jumlah vertices =5, maka ukuran array yang dibuat adalah 6x6, sehingga indeks mulai dari 1 sampai 5 dapat digunakan secara langsung.

3. Apakah kegunaan method **getEdge()** ?
   - Untuk mengembalikan nilai atau bobot edge antara dua node yang ditentukan dan, jika ada.

4. Termasuk jenis graph apakah uji coba pada praktikum 2.2?
   - Menggunkan metode ‘makeEdge’, edge dibuat dengan arah tertentu dari ‘to’ ke ‘from’. Jadi, setiap kali menambahkan edge, memiliki arah yang jelas, merupakan karakteristik dari directed graph.
   - Graph direpresentasikan menggunakan adjacency matrix ‘twoD_array’, di mana nilai ‘twoD_array[to][from]’ diset dengan nilai edge. Ini menunjukkan bahwa ada arah dari vertex ‘to’ ke vertex ‘from’.

5. Mengapa pada method main harus menggunakan *try-catch Exception* ?
   - Karena untuk menangani kemungkinan kesalahan yang bisa terjadi selama eksekusi program. Agar program tidak crash dan dapat memberikan pesan error yang informatif kepada user. Memudahkan dalam debugging karena exception yang dilempar bisa ditangkap dan dilaporkan dengan detail yang lebih baik.

3. Tugas Praktikum

1. Tambahkan scanner pada method AddEdge praktikum 2.1 untuk menerima input Edge

jobsheet12_graph > assignment > GraphMain06.java > ...

```java
public class GraphMain06 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter number of vertices: ");
        int v = sc.nextInt();
        System.out.println("Is the graph directed (true/false): ");
        boolean isDirected = sc.nextBoolean();
        sc.nextLine();
        System.out.println("Enter vertex names: ");
        String[] vertices = new String[v];
        for (int i = 0; i < v; i++) {
            vertices[i] = sc.nextLine();
        }

        Graph06<String> graph = new Graph06<>(vertices, isDirected);

        System.out.println("Enter number of edges: ");
        int e = sc.nextInt();
        sc.nextLine();

        System.out.println("Enter source and destination: ");
        for (int i = 0; i < e; i++) {
            String source = sc.nextLine();
            String destination = sc.nextLine();
            int srcIndex = -1, destIndex = -1;
            for (int j = 0; j < v; j++) {
                if (vertices[j].equals(source)) {
                    srcIndex = j;
                }
                if (vertices[j].equals(destination)) {
                    destIndex = j;
                }
            }
            if (srcIndex != -1 && destIndex != -1) {
                try {
                    graph.addEdge(srcIndex, destIndex);
                } catch (Exception ex) {
                    ex.printStackTrace();
                }
            } else {
                System.out.println("Invalid vertex name");
            }
        }
```

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

```
jobsheet12_graph > assignment > GraphMain06.java > ...
4  public class GraphMain06 {
5      public static void main(String[] args) {
6          Scanner sc = new Scanner(System.in);
7          System.out.println(x:"Enter number of vertices: ");
8          int v = sc.nextInt();
9          System.out.println(x:"Is the graph directed (true/false): ");
10         boolean isDirected = sc.nextBoolean();
11         sc.nextLine();
12         System.out.println(x:"Enter vertex names: ");
13         String[] vertices = new String[v];
14         for (int i = 0; i < v; i++) {
15             vertices[i] = sc.nextLine();
16         }
17
18         Graph06<String> graph = new Graph06<>(vertices, isDirected);
19
20         // Print the graph type
21         if (isDirected) {
22             System.out.println(x:"The graph is directed.");
23         } else {
24             System.out.println(x:"The graph is undirected.");
25         }
```

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

```
jobsheet12_graph > assignment > Graph06.java > ...
3  public class Graph06<T> {
41     public void removeEdge(int source, int destination) throws Exception {
42         list[source].remove(destination);
43         if (!isDirected) {
44             list[destination].remove(source);
45         }
46     }
```

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.

jobsheet12_graph > assignment > J Node06.java > ...

```java
package assignment;

public class Node06 {
    int data;
    Node06 prev, next;

    Node06(Node06 prev, int data, Node06 next) {
        this.prev = prev;
        this.data = data;
        this.next = next;
    }
}
```

jobsheet12_graph > assignment > J DoubleLinkedLists06.java > ...

```java
package assignment;

public class DoubleLinkedLists06 {
    Node06 head;
    int size;

    public DoubleLinkedLists06() {
        head = null;
        size = 0;
    }

    public boolean isEmpty() {
        return head == null;
    }

    public void addFirst(int item) {
        if (isEmpty()) {
            head = new Node06(prev:null, item, next:null);
        } else {
            Node06 newNode = new Node06(prev:null, item, head);
            head.prev = newNode;
            head = newNode;
        }
        size++;
    }

    public void addLast(int item) {
        if (isEmpty()) {
            addFirst(item);
        } else {
            Node06 current = head;
            while (current.next != null) {
                current = current.next;
            }
            Node06 newNode = new Node06(current, item, next:null);
            current.next = newNode;
            size++;
        }
    }
```

jobsheet12_graph > assignment > J DoubleLinkedLists06.java > ...

```
3   public class DoubleLinkedLists06 {
41      public void add(int item, int index) throws Exception {
42          if (index < 0 || index > size) {
43              throw new Exception(message:"Invalid index");
44          }
45          if (index == 0) {
46              addFirst(item);
47              return;
48          }
49          if (index == size) {
50              addLast(item);
51              return;
52          }
53          Node06 current = head;
54          for (int i = 0; i < index - 1; i++) {
55              current = current.next;
56          }
57          Node06 newNode = new Node06(current, item, current.next);
58          current.next.prev = newNode;
59          current.next = newNode;
60          size++;
61      }
62
63      public int size() {
64          return size;
65      }
66
67      public void clear() {
68          head = null;
69          size = 0;
70      }
71
72      public void print() {
73          if (!isEmpty()) {
74              Node06 tmp = head;
75              while (tmp != null) {
76                  System.out.print(tmp.data + "\t");
77                  tmp = tmp.next;
78              }
79              System.out.println(x:"\nSuccessfully added");
80          } else {
81              System.out.println(x:"Linked list is empty");
82          }
83      }
```

jobsheet12_graph > assignment > DoubleLinkedLists06.java > ...

```
3   public class DoubleLinkedLists06 {
85      public void removeFirst() throws Exception {
86          if (isEmpty()) {
87              throw new Exception(message:"Linked list is still empty, can't remove");
88          } else if (size == 1) {
89              head = null;
90          } else {
91              head = head.next;
92              head.prev = null;
93          }
94          size--;
95      }
96
97      public void removeLast() throws Exception {
98          if (isEmpty()) {
99              throw new Exception(message:"Linked list is still empty, can't remove");
100         } else if (head.next == null) {
101             head = null;
102         } else {
103             Node06 current = head;
104             while (current.next != null) {
105                 current = current.next;
106             }
107             current.prev.next = null;
108         }
109         size--;
110     }
111
112     public void remove(int index) throws Exception {
113         if (index < 0 || index >= size) {
114             throw new Exception(message:"Index value is out of bound");
115         }
116         if (index == 0) {
117             removeFirst();
118             return;
119         }
120         if (index == size - 1) {
121             removeLast();
122             return;
123         }
124         Node06 current = head;
125         for (int i = 0; i < index; i++) {
126             current = current.next;
127         }
128         current.prev.next = current.next;
129         current.next.prev = current.prev;
130         size--;
131     }
```

jobsheet12_graph > assignment > J DoubleLinkedLists06.java > ...

```
3   public class DoubleLinkedLists06 {
133     public int getFirst() throws Exception {
134         if (isEmpty()) {
135             throw new Exception(message:"Linked list still empty");
136         }
137         return head.data;
138     }
139
140     public int getLast() throws Exception {
141         if (isEmpty()) {
142             throw new Exception(message:"Linked list still empty");
143         }
144         Node06 tmp = head;
145         while (tmp.next != null) {
146             tmp = tmp.next;
147         }
148         return tmp.data;
149     }
150
151     public int get(int index) throws Exception {
152         if (index < 0 || index >= size) {
153             throw new Exception(message:"Index value is out of bound");
154         }
155         Node06 current = head;
156         for (int i = 0; i < index; i++) {
157             current = current.next;
158         }
159         return current.data;
160     }
161 }
```

jobsheet12_graph > assignment > Graph06.java > ...

```java
package assignment;

public class Graph06<T> {
    T[] vertex;
    DoubleLinkedLists06[] list;
    boolean isDirected;

    public Graph06(T[] vertex, boolean isDirected) {
        this.vertex = vertex;
        this.isDirected = isDirected;
        list = new DoubleLinkedLists06[vertex.length];
        for (int i = 0; i < vertex.length; i++) {
            list[i] = new DoubleLinkedLists06();
        }
    }

    public void addEdge(int source, int destination) throws Exception {
        list[source].addFirst(destination);
        if (!isDirected) {
            list[destination].addFirst(source);
        }
    }

    public void degree(int source) throws Exception {
        System.out.println("Degree of vertex " + vertex[source] + " : " + list[source].size());

        int totalIn = 0, totalOut = list[source].size();
        for (int i = 0; i < vertex.length; i++) {
            for (int j = 0; j < list[i].size(); j++) {
                if ((int) list[i].get(j) == source) {
                    totalIn++;
                }
            }
        }

        System.out.println("Indegree of vertex " + vertex[source] + " : " + totalIn);
        System.out.println("Outdegree of vertex " + vertex[source] + " : " + totalOut);
        System.out.println("Total degree of vertex " + vertex[source] + " : " + (totalIn + totalOut));
    }

    public void removeEdge(int source, int destination) throws Exception {
        list[source].remove(destination);
        if (!isDirected) {
            list[destination].remove(source);
        }
    }
```

jobsheet12_graph > assignment > Graph06.java > ...

```java
public class Graph06<T> {
    public void removeAllEdges() {
        for (int i = 0; i < vertex.length; i++) {
            list[i].clear();
        }
        System.out.println(x:"Graph successfully emptied!");
    }

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

jobsheet12_graph > assignment > GraphMain06.java > GraphMain06 > main(String[])

```java
package assignment;
import java.util.Scanner;

public class GraphMain06 {
    Run | Debug
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println(x:"Enter number of vertices: ");
        int v = sc.nextInt();
        System.out.println(x:"Is the graph directed (true/false): ");
        boolean isDirected = sc.nextBoolean();
        sc.nextLine();
        System.out.println(x:"Enter vertex names: ");
        String[] vertices = new String[v];
        for (int i = 0; i < v; i++) {
            vertices[i] = sc.nextLine();
        }

        Graph06<String> graph = new Graph06<>(vertices, isDirected);

        // Print the graph type
        if (isDirected) {
            System.out.println(x:"The graph is directed.");
        } else {
            System.out.println(x:"The graph is undirected.");
        }

        System.out.println(x:"Enter number of edges: ");
        int e = sc.nextInt();
        sc.nextLine();

        System.out.println(x:"Enter source and destination: ");
        for (int i = 0; i < e; i++) {
            String source = sc.nextLine();
            String destination = sc.nextLine();
            int srcIndex = -1, destIndex = -1;
            for (int j = 0; j < v; j++) {
                if (vertices[j].equals(source)) {
                    srcIndex = j;
                }
                if (vertices[j].equals(destination)) {
                    destIndex = j;
                }
            }
```

jobsheet12_graph > assignment > GraphMain06.java > ...

```
4   public class GraphMain06 {
5       public static void main(String[] args) {
44              if (srcIndex != -1 && destIndex != -1) {
45                  try {
46                      graph.addEdge(srcIndex, destIndex);
47                  } catch (Exception ex) {
48                      ex.printStackTrace();
49                  }
50              } else {
51                  System.out.println(x:"Invalid vertex name");
52              }
53          }
54
55          try {
56              graph.printGraph();
57          } catch (Exception ex) {
58              ex.printStackTrace();
59          }
60
61          System.out.println(x:"Enter a vertex to check degree: ");
62          String vertex = sc.nextLine();
63          int vertexIndex = -1;
64          for (int j = 0; j < v; j++) {
65              if (vertices[j].equals(vertex)) {
66                  vertexIndex = j;
67              }
68          }
69          if (vertexIndex != -1) {
70              try {
71                  graph.degree(vertexIndex);
72              } catch (Exception ex) {
73                  ex.printStackTrace();
74              }
75          } else {
76              System.out.println(x:"Invalid vertex name");
77          }
78          sc.close();
79      }
80  }
```

jobsheet12_graph > assignment > GraphArray06.java > ...

```java
package assignment;

public class GraphArray06<T> {
    private final int vertices;
    private final int[][] twoD_array;
    private T[] vertexLabels;

    public GraphArray06(int v, T[] vertexLabels) {
        vertices = v;
        twoD_array = new int[vertices + 1][vertices + 1];
        this.vertexLabels = vertexLabels;
    }

    public int getVertexIndex(T vertex) {
        for (int i = 0; i < vertexLabels.length; i++) {
            if (vertexLabels[i].equals(vertex)) {
                return i + 1; // shift by 1 to match the adjacency matrix indexing
            }
        }
        return -1; // Vertex not found
    }

    public void makeEdge(T to, T from, int edge) {
        int toIndex = getVertexIndex(to);
        int fromIndex = getVertexIndex(from);

        if (toIndex == -1 || fromIndex == -1) {
            System.out.println(x:"Vertex doesn't exist");
            return;
        }

        try {
            twoD_array[toIndex][fromIndex] = edge;
        } catch (ArrayIndexOutOfBoundsException index) {
            System.out.println(x:"Vertex doesn't exist");
        }
    }

    public int getEdge(T to, T from) {
        int toIndex = getVertexIndex(to);
        int fromIndex = getVertexIndex(from);

        if (toIndex == -1 || fromIndex == -1) {
            System.out.println(x:"Vertex doesn't exist");
            return -1;
        }
```

jobsheet12_graph > assignment > GraphArray06.java > ...

```java
public class GraphArray06<T> {
    public int getEdge(T to, T from) {
        try {
            return twoD_array[toIndex][fromIndex];
        } catch (ArrayIndexOutOfBoundsException index) {
            System.out.println(x:"Vertex doesn't exist");
        }
        return -1;
    }
}
```

jobsheet12_graph > assignment > J GraphArrayMain06.java > ...

```java
package assignment;
import java.util.Scanner;

public class GraphArrayMain06 {
    Run | Debug
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println(x:"Enter number of vertices: ");
        int v = sc.nextInt();
        sc.nextLine();

        System.out.println(x:"Enter vertex names:");
        String[] vertexLabels = new String[v];
        for (int i = 0; i < v; i++) {
            vertexLabels[i] = sc.nextLine();
        }

        GraphArray06<String> graph = new GraphArray06<>(v, vertexLabels);

        System.out.println(x:"Enter number of edges: ");
        int e = sc.nextInt();
        sc.nextLine();

        System.out.println(x:"Insert edges: <to> <from>");
        for (int i = 0; i < e; i++) {
            String to = sc.nextLine();
            String from = sc.nextLine();
            graph.makeEdge(to, from, edge:1);
        }

        System.out.println(x:"2D array as graph representation as follows: ");
        System.out.print(s:"  ");
        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(vertexLabels[i - 1], vertexLabels[j - 1]) + " ");
            }
            System.out.println();
        }

        sc.close();
    }
}
```

--- *** ---