UNIVERSITAS INDONESIA
FAKULTAS ILMU KOMPUTER
Veritas, Probitas, Iustitia

CSGE602040 –
Struktur Data & Algoritma
UAS Semester Gasal 2018/2019
22 Desember 2018, 13.00 – 15.30

Pengajar: Suryana Setiawan, Rahmad Mahendra, Dipta Tanaya, Iis Solichah

---

PERNYATAAN KESANGGUPAN MENTAATI TATA TERTIB UJIAN

"Saya telah membaca dan memahami ketentuan tata tertib berikut ini, serta menyatakan bahwa jawaban ujian ini adalah hasil pekerjaan saya sendiri. Saya menyetujui jika melakukan pelanggaran atas ketentuan tersebut, saya bersedia diproses sesuai ketentuan yang berlaku (SK DGB UI No.1 Tahun 2014) dengan sanksi maksimal **nilai akhir E**."

Nama & Tanda-tangan:	Kelas:	Nomor Pokok Mahasiswa:

TATA TERTIB UJIAN

- Semua alat komunikasi elektronik dalam kondisi non-aktif (dimatikan), dimasukkan ke dalam tas dan diletakkan pada tempat yang telah disediakan.
- Peralatan ujian yang boleh dibawa adalah alat tulis dan yang diperbolehkan sesuai sifat ujian.
- Peserta ujian menempati tempat duduk yang telah ditentukan.
- Peserta ujian menuliskan nama dan NPM pada setiap lembar jawaban ujian.
- Peserta mulai membuka soal dan mengerjakan ketika pengawas mengatakan ujian dimulai dan berhenti bekerja (meletakkan alat tulis) ketika pengawas mengatakan waktu habis.
- Peserta tidak berkomunikasi dalam bentuk apa pun dengan peserta lain selama berada di ruang ujian, termasuk pinjam meminjam alat tulis, serta tidak memberi atau menerima bantuan dari siapapun selama ujian berlangsung.
- Peserta yang meninggalkan ruang ujian dianggap selesai mengerjakan. Jika karena kondisi medis khusus tidak bisa memenuhi ketentuan ini, peserta wajib melaporkan kepada pengawas sebelum ujian dimulai.
- Setelah selesai mengerjakan atau setelah waktu habis, peserta segera meninggalkan berkas soal dan lembar jawaban ujian di meja masing-masing, mengambil tas dan segera keluar tanpa mengganggu peserta lain serta tanpa berkomunikasi dengan peserta lain.
- Jawaban ujian ini tidak akan dinilai jika pernyataan di atas ini tidak ditandatangani.

Informasi Tambahan

- Jawaban ditulis pada lembar jawaban yang disediakan
- Berdasarkan aturan yang disebutkan di BRP, nilai UAS dari mahasiswa yang persentase kehadiran (perkuliahan di kelas dan sesi lab) < 65% tidak akan dihitung dalam penilaian akhir.

Nama		Kelas	
NPM		Nomor Meja	

Lembar Jawaban Ujian Akhir Semester
Struktur Data dan Algoritma
Semester Gasal 2018-2019

Petunjuk Umum:

- *Tulis nama, NPM, kelas, dan nomor meja Anda di setiap halaman lembar jawaban*
- *Nilai total: 110 poin*
- *Baca soal dengan **teliti** dan tulis jawaban Anda dengan tulisan yang **jelas** pada **halaman lembar jawaban***

Bagian A

Nomor	**Jawaban**
1	
2	
3	
4	
5	
6	
7	
8	
9	

Nama		Kelas	
NPM		Nomor Meja	

10

11

12

13

14

Indeks	0	1	2	3	4	5	6	7	8	9	10	11	12
Data													

15

Indeks	0	1	2	3	4	5	6	7	8	9	10	11	12
Data													

16

Indeks	Key
0	
1	
2	
3	
4	
5	
6	

17

Indeks	Key
0	
1	
2	
3	
4	
5	
6	

18

19

20

Nama		Kelas	
NPM		Nomor Meja	

Bagian B

No	Jawaban
1	
2	

Nama		Kelas	
NPM		Nomor Meja	

3

Vertex	Jarak ke vertex A
A	
B	
C	
D	
E	
F	
G	

4

Space tambahan untuk jawaban:

Nama		Kelas	
NPM		Nomor Meja	

Bagian C

Nomor 1

```
class Graph{
	private static int JUMLAH_VERTEX;
	private static boolean[][] adjacencyMatrix;
	static int[] inDegree;

	Graph(int v){
		JUMLAH_VERTEX = v;
		adjacencyMatrix = new boolean[v][v];
		inDegree = new int[v];
	}

	void addEdge(int i, int j){
		adjacencyMatrix[i][j] = true;
	}

	public static void main(String[] args){
		Graph g = new Graph(3);
		g.addEdge(0,1);
		g.addEdge(0,2);
		g.addEdge(2,1);

		g.countInDegree();

		for(int i:inDegree){
			System.out.println(i);
		}
	}

	void countInDegree()	// lengkapi method ini









}
```

Nama		Kelas	
NPM		Nomor Meja	

Nomor 2

```
class BinaryNode{
    int element;
    BinaryNode left;
    BinaryNode right;

    public BinaryNode(int e, BinaryNode left, BinaryNode right){
        element = e;
        this.left = left;
        this.right = right;
    }
}

public class BinaryTree{
    BinaryNode root;
    public BinaryTree(BinaryNode r){
        root = r;
    }

    public Boolean isAVL(BinaryNode root)      // implementasikan solusi Anda

}
```

Nama		Kelas	
NPM		Nomor Meja	

Nomor 3

```
class Graph{
    private static int JUMLAH_VERTEX;
    private static ArrayList<Integer> adjList[];

    Graph(int v){
        JUMLAH_VERTEX = v;
        adjList = new ArrayList[JUMLAH_VERTEX];
        for (int i=0; i<JUMLAH_VERTEX; i++)
            adjList[i] = new ArrayList();
    }

    void addEdge(int v, int w){
        adjList[v].add(w);
        adjList[w].add(v);
    }

    public static void main(String args[]) throws IOException{
          BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
          String[] in = reader.readLine().split(" ");
          int jmlVertex = Integer.parseInt(in[0]);
          int jmlEdge = Integer.parseInt(in[1]);

          Graph graf = new Graph(jmlVertex);
          for(int i=0; i<jmlEdge; i++){
               in = reader.readLine().split(" ");
               graf.addEdge(Integer.parseInt(in[0]), Integer.parseInt(in[1]));
          }

          in = reader.readLine().split(" ");
          int from = Integer.parseInt(in[0]);
          int to = Integer.parseInt(in[0]);
          printSimplePath(from, to);
    }

//implementasikan solusi Anda di sini
```

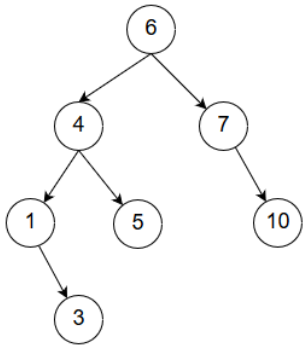
Nama		Kelas	
NPM		Nomor Meja	

}

Lembar Soal Ujian Akhir Semester
Struktur Data dan Algoritma
Semester Gasal 2018/2019

Bagian A
(20 soal @ 2.5 poin. Nilai total: 50 poin)

1. Perhatikan *tree* berikut ini



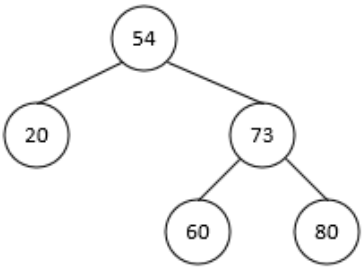
   Jika elemen dari *binary tree* dicetak secara *in-order traversal*, urutan pencetakannya adalah …

2. Sebuah BST yang balanced memiliki *k* elemen, berapa kompleksitas operasi *insert* elemen baru ke dalam BST tersebut?

3. Sebuah *binary tree* yang memiliki 101 *nodes*. Jika tidak ada *internal node* yang jumlah *child*-nya 1, berapa banyak *leaf node* dalam *binary tree* tersebut?

4. Kasus terburuk operasi-operasi pada Binary Search Tree dengan N buah *node* terjadi ketika:
   a. Tinggi *tree* sama dengan N log N
   b. Tinggi *tree* sama dengan log N
   c. Tinggi *tree* sama dengan jumlah *node*
   d. Tree merupakan *complete binary tree*
   e. N adalah bilangan yang sangat besar

Soal 5 dan 6 mengacu pada AVL tree berikut.

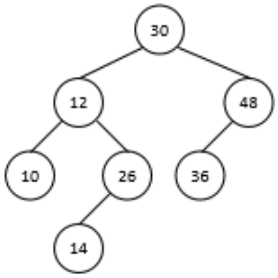


5. Gambarkan *tree* setelah operasi *insert* 57

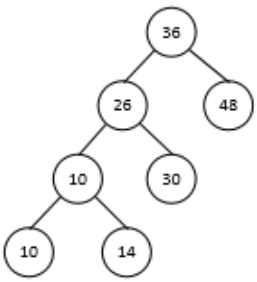
6. Gambarkan *tree* setelah operasi *delete* 20
   (soal no 6 ini mengacu pada AVL Tree awal dan tidak memperhitungkan operasi pada soal no 5)

7. Jika dilakukan operasi *insertion* pada AVL Tree dengan urutan data: 48, 36, 26, 14, 30, 10, 12.
   AVL Tree yang terbentuk adalah:

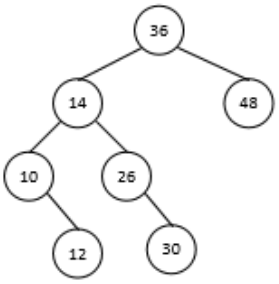
a.



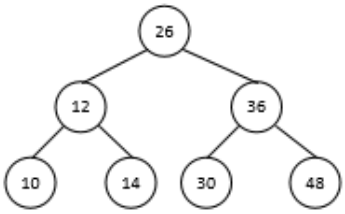
b.



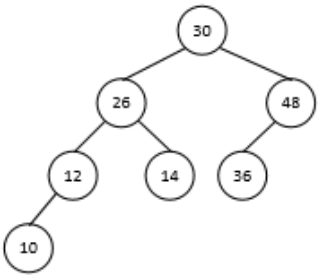
c.



d.



e.



8. Berapa jumlah *node* minimum pada AVL Tree dengan tinggi 6?

9. Berapa kompleksitas operasi rotasi setelah dilakukan *insertion* sebuah data pada AVL Tree tanpa memperhitungkan kompleksitas proses pencarian *node* yang perlu dirotasi?

10. Diberikan data berikut: 233, 205, 832, 421, 849, 724, 899 yang dimasukkan ke B+Tree yang awalnya masih kosong.
    Diketahui *max-degree* (*maximum branching factor*) dari B+ tree tersebut adalah 5. Operasi *split* dilakukan dengan membagi 3 *keys* di sebelah kiri dan 2 *keys* di sebelah kanan.
    Gambarkan B+Tree yang dihasilkan.

11. Data sejumlah 1.000.000 *entries* disimpan dalam sebuah B+tree dengan *max-degree* (*maximum branching factor*) = 5.
    Jika tinggi B+tree tersebut adalah *t*, maka nilai *t* yang mungkin adalah …… ≤ *t* ≤ …….
    Ingat: Node pada B+tree yang mendekati kosong akan dimerge atau diredistribute

12. Berapa kompleksitas algoritma *heap sort* untuk mengurutkan data yang disimpan pada *full binary tree* dengan tinggi *h*?

13. *N* elemen berbeda akan disimpan ke sebuah hash table. Masalah **second clustering** kemungkinan bisa terjadi jika hash table menerapkan mekanisme collision resolution berikut
    (jawaban bisa lebih dari satu)

    a. Linear probing
    b. Quadratic probing
    c. Double hash
    d. Open hashing
    e. Chaining

Soal nomor 14 dan 15 mengacu pada array berikut yang merupakan representasi minimum binary heap.

Indeks	0	1	2	3	4	5	6	7	8	9	10	11	12
Data	5	10	20	12	17	24	30	19	16	21	18		

14. Tambahkan data 9 dan 13 ke dalam *heaptree* tersebut dan tuliskan isi *array*.

15. Berdasarkan *heaptree* **awal** di atas (operasi pada soal no 14 tidak diperhitungkan), hapus dua data terkecil dan tuliskan isi *array*.

Soal nomor 16 dan 17 mengacu pada tabel di bawah ini.

Diberikan sejumlah *key* dan fungsi *hash* dari *key* tersebut sebagai berikut.

Key	***H(Key)***
Sumatera	75
Jawa	27
Kalimantan	36
Bali	19
Sulawesi	56

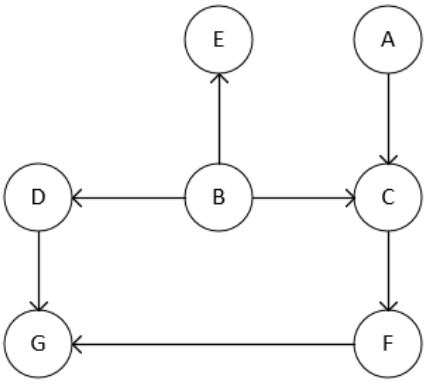
Ukuran tabel *hash* adalah 7 dengan indeks dari 0 sampai dengan 6.

16. Bagaimana isi tabel *hash* setelah seluruh data masuk ke dalam table jika *collition* ditangani dengan ***linear probing***?

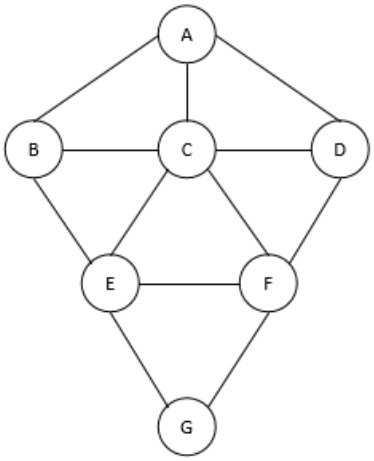
17. Bagaimana isi tabel *hash* setelah seluruh data masuk ke dalam table jika *collition* ditangani dengan ***quadratic probing***?

18. Perhatikan graf di bawah ini. Tentukan hasil *topological sorting* dari graf tersebut (jawaban bisa lebih dari satu):

    a. A, B, C, D, E, F, G
    b. A, B, C, D, E, G, F
    c. B, D, A, C, E, F, G
    d. B, D, E, C, A, F, G
    e. B, E, A, C, D, F, G



Soal 19 dan 20 mengacu pada graf berikut

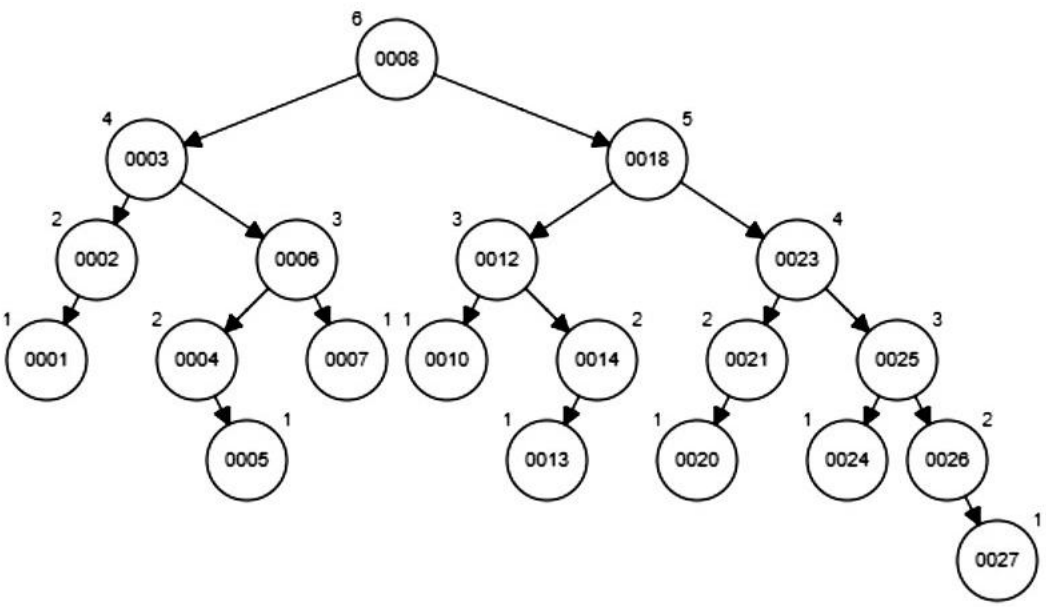


19. Jika algoritma BFS dijalankan pada graf tersebut dimulai dari vertex D, manakah urutan *traversal* yang salah?
    a. D, A, C, F, B, E, G
    b. D, A, C, B, E, F, G
    c. D, A, C, F, B, G, E
    d. D, F, C, A, B, E, G
    e. D, F, A, C, G, E, B

20. Jika algoritma DFS dijalankan pada graf tersebut dimulai dari vertex F dan saat menemukan percabangan, cabang dengan *node* yang secara leksikografis lebih kecil akan dipilih terlebih dahulu, Tuliskan urutan *traversal*-nya

Bagian B

(4 soal. Nilai total: 30 poin)



1. (Nilai: 6 poin)
   Lakukan penghapusan *key* = 003 pada AVL Tree di atas dengan menggunakan *predecessor inorder*. Gambarkan *tree* setelah penghapusan dilakukan.

2. [8 poin]

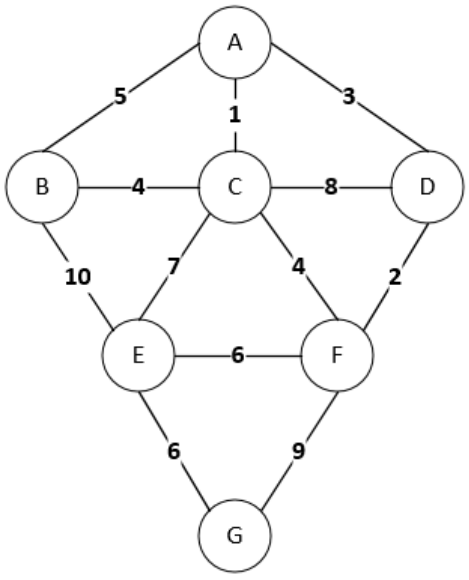
Diberikan sebuah potongan *code* sebagai berikut. *Method* fun1 menerima parameter *input* BinaryNode *t* yang merupakan *root* dari sebuah Binary Search Tree (BST).

```
BinaryNode fun1 (BinaryNode t){
    if (t == null) throw exception;
    while (t.right != null) {
        t = t.right;
    }
    return t;
}
```

a. Jelaskan dengan singkat apa yang dilakukan *method* **fun1** tersebut?

b. Diberikan data String sebagai berikut: “Sumatera”, “Jawa”, “Kalimantan”, “Bali”, “Sulawesi”, “Madura”, “Lombok”, “Sumbawa”.
Gambarkan visualisasi *tree* BST yang menyimpan seluruh data tersebut di mana eksekusi *method* **fun1** merupakan kasus *worst case.*

c. Tentukan urutan *insertion* data String tersebut ke dalam BST sehingga menghasilkan *tree* seperti jawaban no b.



Gambar Graph Berbobot

3. [6 poin]
Terapkan algoritma Djikstra pada graf berbobot di atas untuk mencari jarak terpendek dari vertex A ke vertex E. Tuliskan isi tabel Djikstra (lihat lembar jawaban) untuk masing-masing vertex ketika vertex E sudah diketahui jarak minimumnya dari A (saat E masuk ke dalam “*white-cloud*”).

4. [10 poin]

Pelajari *code* Java berikut

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
import java.util.Iterator;
import java.util.List;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Map;
import java.util.HashMap;
import java.util.Queue;
import java.util.PriorityQueue;


class Vertex {
    String label;

    Vertex(String name) {
           label = name;
    }
    public boolean sameVertex(Vertex anotherVertex) {
           return this.label.equals(anotherVertex.label);
    }
}


class Edge implements Comparable<Edge> {
    Vertex v1;
    Vertex v2;
    int weight;

    Edge(Vertex v1, Vertex v2, int weight) {
           this.v1 = v1;
           this.v2 = v2;
           this.weight = weight;
    }

    public void printEdge() {
           System.out.println(v1.label + " " + v2.label);
    }

    public int compareTo(Edge anotherEdge){
           return Integer.valueOf(this.weight).compareTo
                (Integer.valueOf(anotherEdge.weight));
    }
}
```

```
class GraphAnalysis{

    public static void main(String args[]) throws IOException{
          BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
          String[] in = reader.readLine().split(" ");
          int numOfVertices = Integer.parseInt(in[0]);
          int numOfEdges = Integer.parseInt(in[1]);

          List<Vertex> allVertices = new ArrayList<Vertex>();
          List<Edge> allEdges = new ArrayList<Edge>();

          for(int i=0; i<numOfVertices; i++){
                String vertexLabel = reader.readLine();
                allVertices.add(new Vertex(vertexLabel));
          }

          for(int j=0; j<numOfEdges; j++){
                 in = reader.readLine().split(" ");
                 Vertex firstVertex = findVertexByLabel(in[0], allVertices);
                 Vertex secondVertex = findVertexByLabel(in[1], allVertices);
                 int thisIsWeight = Integer.parseInt(in[2]);

                 Edge aNewEdge = new Edge(firstVertex, secondVertex, thisIsWeight);
                 allEdges.add(aNewEdge);
          }

          Graph thisIsGraph = new Graph(allVertices, allEdges);

          Graph findThisGraphForYourMark = thisIsGraph.anotherGraph(true);
          System.out.println("Graph TRUE: ");

          for(int k=0; k<findThisGraphForYourMark.edges.size(); k++) {
                 findThisGraphForYourMark.edges.get(k).printEdge();
          }

          Graph thisGraphForAnotherMark = thisIsGraph.anotherGraph(false);
          System.out.println("\nGraph FALSE: ");

          for(int k=0; k<thisGraphForAnotherMark.edges.size(); k++) {
                 thisGraphForAnotherMark.edges.get(k).printEdge();
          }

          int luckyNumber = thisGraphForAnotherMark.countUsingBFS();
          System.out.println("\n" + luckyNumber);
    }


    public static Vertex findVertexByLabel(String name, List<Vertex> vertices) {
          for(int i=0; i<vertices.size(); i++) {
                 Vertex thisIsVertex = vertices.get(i);
                 if(thisIsVertex.label.equals(name))
                        return thisIsVertex;
          }
          return new Vertex(name);
    }
}
```

```
class Graph {

    List<Vertex> vertices;
    List<Edge> edges;

    Graph(List<Vertex> listOfVertices, List<Edge> listOfEdges) {
           vertices = listOfVertices;
           edges = listOfEdges;
           adjacencyList = adjacents();
    }

    Map<Vertex, List<Vertex>> adjacents(List<Edge> listOfEdges) {
           Map<Vertex, List<Vertex>> newAdjacents = new HashMap<> ();
           List<Vertex> adj;

           for(int b = 0; b < listOfEdges.size(); b++) {
                  Edge connection = listOfEdges.get(b);

                  if(newAdjacents.containsKey(connection.v1))
                         adj = newAdjacents.get(connection.v1);
                  else
                         adj = new ArrayList<Vertex>();

                  adj.add(connection.v2);
                  newAdjacents.put(connection.v1, adj);

                  if(newAdjacents.containsKey(connection.v2))
                         adj = newAdjacents.get(connection.v2);
                  else
                         adj = new ArrayList<Vertex>();

                  adj.add(connection.v1);
                  newAdjacents.put(connection.v2, adj);
           }
           return newAdjacents;
    }

    Graph anotherGraph(boolean typeOfGraph){
           Queue<Edge> myQueue = new PriorityQueue<Edge>();

           for(int i = 0; i < edges.size(); i++){
                  myQueue.add(edges.get(i));
           }

           List<Edge> edgeMainGroup = new ArrayList<Edge>();
           List<Edge> edgeEliminatedGroup = new ArrayList<Edge>();

           while(!myQueue.isEmpty()){
                  Edge thisIsAnEdge = myQueue.poll();
                  if(edgeMainGroup.size() < this.vertices.size() - 1) {
                         edgeMainGroup.add(thisIsAnEdge);
                         if(containsDonut(thisIsAnEdge, edgeMainGroup)) {
                                edgeMainGroup.remove(thisIsAnEdge);
                         }
                  }
                  else
                         edgeEliminatedGroup.add(thisIsAnEdge);
           }

           if(typeOfGraph)
                  return new Graph(vertices, edgeMainGroup);
           return new Graph(vertices, edgeEliminatedGroup);
    }
    int countUsingBFS(){
```

```
        Map<Vertex, List<Vertex>> adjacencyList = adjacents(this.edges);
         Map<Vertex, Boolean> visited = new HashMap<> ();

        for(int a = 0; a < vertices.size(); a++) {
            visited.put(vertices.get(a), new Boolean(false));
        }

        //LinkedList "visitQueue" adalah implementasi sebuah queue
        LinkedList<Vertex> visitQueue = new LinkedList();
        Iterator<Vertex> itrSequence = visitQueue.listIterator();
        int counter = 0;

        for(int i = 0; i < vertices.size(); i++){
            Vertex oneNode = vertices.get(i);

            if(visited.get(oneNode).booleanValue() == false) {
                  counter++;
                  visitQueue.add(oneNode);
                  visited.put(oneNode, new Boolean(true));

                  while(itrSequence.hasNext()){
                        Vertex dequeuedNode = visitQueue.poll();
                        if(adjacencyList.containsKey(dequeuedNode)) {
                               List<Vertex> ourNeighbors = adjacencyList.get(dequeuedNode);
                               Iterator<Vertex> itr = ourNeighbors.listIterator();

                               while(itr.hasNext()){
                                   Vertex oneNeighbor = itr.next();
                                   if(visited.get(oneNeighbor).booleanValue() == false) {
                                          visited.put(oneNeighbor, new Boolean(true));
                                          visitQueue.add(oneNeighbor);
                                   }
                              }
                         }
                   }
            }
        }

        return counter;
    }


    boolean checkDonutWithDFS(Vertex current, Vertex previous, Map<Vertex, List<Vertex>>
adjacencies, Map<Vertex, Boolean> visited) {
         boolean res = false;
         visited.put(current, new Boolean(true));
         Iterator<Vertex> vertexIter = adjacencies.get(current).listIterator();

         while (vertexIter.hasNext()){
            Vertex next = vertexIter.next();
            if(visited.get(next).booleanValue() == false){
                  if(checkDonutWithDFS(next, current, adjacencies, visited))
                      return true;
            }
            else if(!next.sameVertex(previous)){
                  return true;
            }
         }
         return false;
    }

    boolean containsDonut(Edge singleEdge, List<Edge> edgeGroup) {
```

```
            Map<Vertex, List<Vertex>> listOfAdjacencies = adjacents(edgeGroup);
            Map<Vertex, Boolean> visited = new HashMap<> ();
            for(int a = 0; a < vertices.size(); a++) {
                   visited.put(vertices.get(a), new Boolean(false));
            }

            //parameter kedua adalah vertex yang dijamin bukan merupakan vertex dari Graph
            if(checkDonutWithDFS(singleEdge.v1, new Vertex("#"), listOfAdjacencies, visited))
                   return true;

            return false;
     }

} //akhir dari class Graph
```

a. Method **anotherGraph** pada code di atas (halaman 16) adalah implementasi dari modifikasi algoritma …….

b. Tuliskah output program jika method **main** pada class **GraphAnalysis** (halaman 15) menerima input sebagai berikut

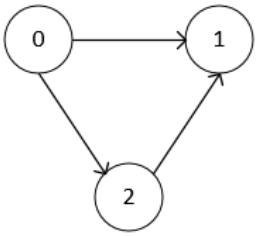
```
7 12
A
B
C
D
E
F
G
A B 5
A C 1
A D 3
B C 4
B E 10
C D 8
C E 7
C F 4
D F 2
E F 6
E G 6
F G 9
```

Keterangan: input yang diberikan soal 4b ini adalah graf berbobot yang sama dengan Gambar soal 3 (halaman 13)

Bagian C
(3 soal @10 poin. Nilai total: 30 poin)

1. Diberikan *directed graph* dalam representasi *adjacency matrix* dengan nilai elemen matriks [i,j] *true* jika vertex **i** memiliki *adjacent* ke vertex **j**. Untuk menjalankan algoritma Topological Sort, diperlukan indegree dari setiap vertex.
   Buatlah sebuah *method* untuk menghitung *in-degree* dari setiap vertex.

   Perhatikan contoh representasi matriks graf berikut



		Vertex **j**		
		0	1	2
Vertex **i**	0	false	true	true
	1	false	false	false
	2	false	true	false

2. Buatlah sebuah *method* yang menerima parameter *input* sebuah **Tree** dan mengembalikan nilai *boolean* apakah Tree tersebut merupakan **AVL Tree** atau bukan.

   Untuk soal no 2 ini, *input* tree yang akan diuji diasumsikan sudah terdefinisi sebagai object dari Class BinaryTree (lihat template jawaban).
   Anda tidak diminta mengimplementasikan method main untuk memroses input.

3. Diberikan sebuah ***undirected graph.*** Buatlah sebuah *method* (menerima parameter ***input*** **dua vertex** A dan B) yang **mencetak seluruh lintasan elementer** dari vertex A ke vertex B.
   Elementary path adalah path yang <u>TIDAK mengulang *vertex*</u>

   <u>Format input</u>:
   Terdapat (*e* + 2) baris input
   - Baris pertama berisi dua bilangan *v* dan *e*. *v* menyatakan jumlah vertex (dalam soal ini, himpunan vertex adalah bilangan bulat dari 0 sampai *v* – 1). *e* menyatakan jumlah edge
   - *e* baris berikutnya yang mendaftarkan himpunan *edge*
   - Baris terakhir berisi dua bilangan *m* dan *n*. Soal yang ditanyakan berarti: cetaklah seluruh *elementary path* dari vertex *m* ke *vertex n*

   <u>Format output</u>:
   Elementary path dicetak per baris (tidak ada ketentuan urutan pencetakan)

Contoh input output:
Misalnya diberikan *complete graph* yang terdiri dari 4 vertex

Contoh Format Input:

```
4 6
0 1
0 2
0 3
1 2
1 3
2 3
1 3
```

Yang ditanyakan adalah cetak seluruh *elementary path* dari vertex 1 ke vertex 3.

Salah satu Kemungkinan Output yang Benar:

(karena urutan pencetakan tidak ada ketentuan khusus, Anda cukup mengimplementasikan solusi untuk salah satu kemungkinan saja)

```
1 3
1 0 3
1 2 3
1 0 2 3
1 2 0 3
```