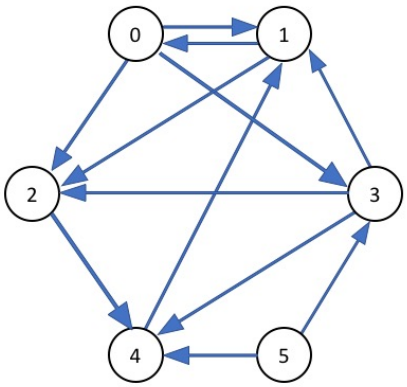
Exercise 3. Answer Sheet

Student's Name: Yuta Nemoto Student's ID: s1240234

Problem 1. *(25 points)* Consider the following adjacency matrix:

0 1 1 1 0 0
1 0 1 0 0 0
0 0 0 0 1 0
0 1 1 0 1 0
0 1 0 0 0 0
0 0 0 1 1 0

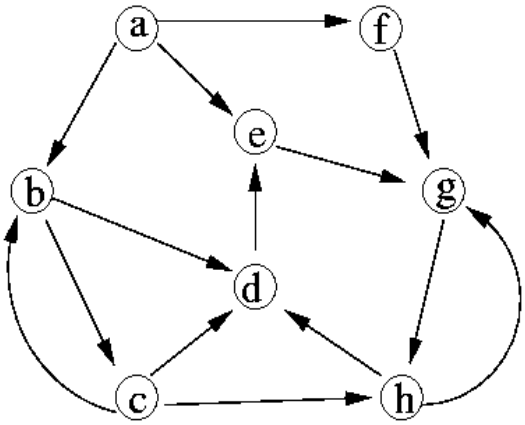
a) Draw a directed graph which corresponds to this adjacency matrix.



b) Write the adjacency list of the graph from a).

0 → 1, 2, 3
1 → 0, 2
2 → 4
3 → 1, 2, 4
4 → 1
5 → 3, 4

Problem 2. *(25 points)* Consider the following graph:



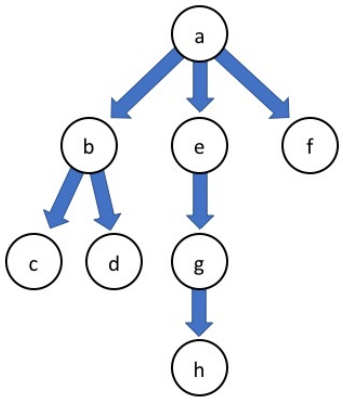
a) Starting from vertex a, in what order the Breath First Search algorithm will traverse the vertices of this graph?

First,
the nodes at distance 1 from a are **b**, **e**, **f**
the nodes at distance 2 from a are **c**, **d**, **g**
the node at distance 3 from a is **h**.

In Breadth-First Search algorithm, it visits other vertices at increasing distances away from a, so the order is
b, e, f → c, d, g → h

b) Starting from vertex a, in what order the Depth First Search algorithm will traverse the vertices of this graph?

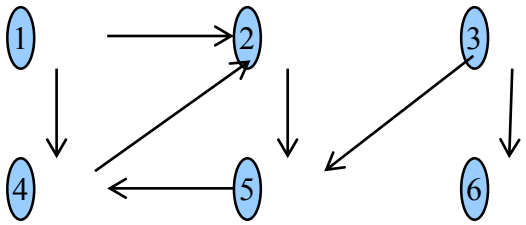
First, the tree generated by the graph is presented below.



In Depth-First Search algorithm, it explores a path all the way to a leaf before backtracking and exploring another path, so the order is
a, b, c, d, e, g, h, f

Problem 3. *(50 points)* Based on the pseudo-code for Depth First Search algorithm given at the lecture, write a program implementing it. Given the following graph, starting from node 1, calculate the discovery

and finishing time of each node and fill the table starting from node 1. (Don't forget to upload your program!)



Node	1	2	3	4	5	6
Discovery time	1	2	9	4	3	10
Finishing time	8	7	12	5	6	11

<Actual Interface>

```
bash-3.2$ javac DirectedEdge.java
bash-3.2$ javac Node.java
bash-3.2$ javac Graph.java
bash-3.2$ java Graph
Please enter the number of nodes in the graph:
6
Please enter the adjacency list of this graph
(Elements must be INTEGER, and must be separated by space. Input any alphabet to finish input.):
1 -> 2 4 end
2 -> 5 end
3 -> 5 6 end
4 -> 2 end
5 -> 4 end
6 -> end
Node            1       2       3       4       5       6
Discovery time  1       2       9       4       3       10
Finishing time  8       7       12      5       6       11
bash-3.2$ java Graph
Please enter the number of nodes in the graph:
17
Please enter the adjacency list of this graph
(Elements must be INTEGER, and must be separated by space. Input any alphabet to finish input.):
1 -> 2 3 end
2 -> 4 5 end
3 -> 6 7 end
4 -> end
5 -> 8 9 end
6 -> end
7 -> 10 11 end
8 -> 12 13 14 end
9 -> 15 16 end
10 -> end
11 -> 17 end
12 -> end
13 -> end
14 -> end
15 -> end
16 -> end
17 -> end
Node            1       2       3       4       5       6       7       8       9       10      11      12      13      14      15      16      17
Discovery time  1       2       22      3       5       23      25      6       14      26      28      7       9       11      15      17      29
Finishing time  34      21      33      4       20      24      32      13      19      27      31      8       10      12      16      18      30
```

<Output result>

```
Node            1       2       3       4       5       6
Discovery time  1       2       9       4       3       10
Finishing time  8       7       12      5       6       11
```

<How to compile/run>

Command: **javac DirectedEdge.java**
javac Node.java
javac Graph.java

java Graph

Input: First, you need to input the number of nodes in the graph.
After that, the numbered nodes are generated automatically and you need to input the adjacency list of the graph.
When you input the data of adjacency list, please enter the numbers separated by space. And after you finished this operation, please input any alphabet character(In the example above, it's the string "end") to end the input operation.

<Source Code>

Graph.java

```
import java.util.ArrayList;
import java.util.Scanner;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Graph {

	// ArrayList of the Node in this Graph
	private ArrayList<Node> nodes = new ArrayList<Node>();
	// Time Parameter for Searching
	private int time;

	/* Constructor */
	public Graph() { }

	/* Add the element of this Graph */
	public void addNode(int number) {
		this.nodes.add(new Node(number));
	}

	/* DFS Method (This time, this method explore the graph from the node '1' by default) */
	public void DFS(Graph g){
		// Initialization (The color of each vertex(Node) is WHITE as default)
		this.time = 0;
		// Visit each node
		for(int i = 0; i < this.nodes.size(); i++)
			if(this.nodes.get(i).getColor() == Node.WHITE)
				this.DFS_VISIT(g, nodes.get(i));
	}

	/* DFS-VISIT Method */
	public void DFS_VISIT(Graph g, Node u) {
		// Update the time variable
		time++;
		// Put the discovery time to the Node
		u.setDiscoveredTime(time);
		// Discover the Node u and set the color of this to GRAY
		u.setColor(Node.GRAY);
		// If the next Node of u is still not discovered, visit the node.
		for(int i = 0; i < u.getDegree(); i++)
			if(u.getDest(i).getColor() == Node.WHITE)
				this.DFS_VISIT(g, u.getDest(i));
		// Finish the exploration of the Node u
		u.setColor(Node.BLACK);
		// Update the time variable
		time++;
		// Put the finished time to the Node
		u.setFinishedTime(time);
	}

	/* Require the adjacency list from the standard input and setup the Graph. */
	public void setup() {
		// Prepare Scanner for input
		Scanner scan = new Scanner(System.in);

		// Output the message
		System.out.println("Please enter the number of nodes in the graph:");
		// Get the number of nodes in the Graph
		int nodeNum = scan.nextInt();
		// Generate the Node in this Graph
		for(int i = 1; i <= nodeNum; i++) this.addNode(i);

		// Output the message
		System.out.println("Please enter the adjacency list of this graph \n(Elements must be INTEGER, and must be sepa-
rated by space. Input any alphabet to finish input.):");
		// Loop for the input
		for(int i = 0; i < nodeNum; i++) {
			System.out.print((i + 1) + " -> ");
			int[] input = new int[nodeNum];
			int j;
			for(j = 0; j < nodeNum; j++) {
```

```
                                        // Read the inputed elements
                                        String temp = scan.next();
                                        // Set the acceptable pattern by the regular expression. Minus value is also acceptable.
                                        Pattern p = Pattern.compile("^-?[0-9]*$");
                                        // Check whether it matches to the pattern p or not.
                                        Matcher m = p.matcher(temp);
                                        // If the input is integer value, add it to the array. Else, end the loop.
                                        if(m.find()) {
                                                // If the inputed node doesn't exist, output an error.
                                                int tempInt = Integer.valueOf(temp);
                                                if(tempInt < 1 || tempInt > nodeNum) {
                                                        System.err.println("Node " + tempInt + " doesn't exist!");
                                                        break;
                                                }
                                                // if it's exist, put it in the array of input.
                                                input[j] = tempInt;
                                        }
                                        else break;
                                }
                                // Create the directed edge to the Nodes in this Graph by following the inputed data
                                for(int k = 0; k < j; k++) {
                                        this.nodes.get(i).addConnectionTo(this.nodes.get(input[k] - 1));
                                }
//                              for(; j >= 0; j--) this.nodes.get(i).addConnectionTo(this.nodes.get(input[j] - 1));
                        }
                }

                /* Output the result */
                public void outResult() {
                        System.out.print("Node\t\t");
                        for(int i = 0; i < this.nodes.size(); i++) System.out.print(this.nodes.get(i).getNumber() + "\t");
                        System.out.println();
                        System.out.print("Discovery time\t");
                        for(int i = 0; i < this.nodes.size(); i++) System.out.print(this.nodes.get(i).getDiscoveredTime() + "\t");
                        System.out.println();
                        System.out.print("Finishing time\t");
                        for(int i = 0; i < this.nodes.size(); i++) System.out.print(this.nodes.get(i).getFinishedTime() + "\t");
                        System.out.println();
                }

                /* Main Method */
                public static void main(String[] args) {
                        // Declare the Graph object
                        Graph graph = new Graph();
                        // Require input
                        graph.setup();
                        // Execute the Depth-First Searching
                        graph.DFS(graph);
                        // Output the result
                        graph.outResult();
                }

}
```

Node.java

```
import java.util.ArrayList;

public class Node {

        // Defined variable for searching in the graph(Judge this node is already searched or not by using this code)
        public static final int WHITE = 0; // WHITE means it's unvisited Node while this search
        public static final int GRAY = 1;  // GRAY means it's discovered Node while this search
        public static final int BLACK = 2; // BLACK means it's finished Node while this search

        // The number of this node
        private int number;
        // Direction pointer to other Node
        private ArrayList<DirectedEdge> directionPointer = new ArrayList<DirectedEdge>();
        // Variable for searching in the graph(It has the value WHITE, GRAY, or BLACK of this static integer
        private int color;
        // Variable of Time Stamp(Initialized by -1)
        private int discoveredTime = -1;
        private int finishedTime = -1;

        /* Constructor */
        public Node(int number) {
                this.number = number;
        };

        /* Add a new connection to the other Node */
        public void addConnectionTo(Node toPoint) {
                this.directionPointer.add(new DirectedEdge(this, toPoint));
        }

        /* Set the color of this node */
        public void setColor(int color) {
```

```
                switch(color) {
                case Node.WHITE:
                        this.color = Node.WHITE;
                        break;
                case Node.GRAY:
                        this.color = Node.GRAY;
                        break;
                case Node.BLACK:
                        this.color = Node.BLACK;
                        break;
                default:
                        System.err.println("Wrong argument is given to Node(" + this.number + ").changeColor(). Please give the
integer WHITE, GRAY, or BLACK.");
                        this.color = Node.WHITE;
                        break;
                }
        }

        /* Get the number of this node */
        public int getNumber() {
                return this.number;
        }

        /* Get the color of this node */
        public int getColor() {
                return this.color;
        }

        /* Get the degree of this node */
        public int getDegree() {
                return this.directionPointer.size();
        }

        /* Get the destination of the direction edge of this node by the index */
        public Node getDest(int index) {
                return this.directionPointer.get(index).getDestination();
        }

        /* Set the discovered time in this search */
        public void setDiscoveredTime(int time) {
                this.discoveredTime = time;
        }

        /* Get the discovered time in this search */
        public int getDiscoveredTime() {
                return this.discoveredTime;
        }

        /* Set the finished time in this search */
        public void setFinishedTime(int time) {
                this.finishedTime = time;
        }

        /* Get the finished time in this search */
        public int getFinishedTime() {
                return this.finishedTime;
        }
}
```

DirectedEdge.java

```
public class DirectedEdge {
        // The Origin of this Edge
        private Node origin;
        // The Destination of this Edge
        private Node destination;

        /* Constructor */
        public DirectedEdge(Node from, Node to) {
                this.origin = from;
                this.destination = to;
        }

        /* Return the origin of this Edge */
        public Node getOrigin() {
                return this.origin;
        }

        /* Return the destination of this Edge */
        public Node getDestination() {
                return this.destination;
        }
}
```