COMP 7038

NON-LINEAR DATA STRUCTURES & ALGORITHMS

ASSIGNMENT 1 – Graphs & trees

SUBMISSION & DUE DATE

This assignment should be submitted to Canvas before 11:59pm on **Friday 7/3/2025**. The usual late submission penalties apply in accordance with MTU's marks and standards policy.

Please submit a single ZIP file with your student number and name in the filename. Your submission should contain **exactly 3 files**:

- *GraphAdjMatrix.java*
- *GraphAdjList.java*
- *BinarySearchTreeImpl.java*

Please do NOT include any other files in your submission.

Make sure that you document your code and include sufficient comments where appropriate. Explain exactly what you are implementing and provide arguments for the correctness of your solution.

EVALUATION PROCEDURE

You can achieve a total of 25 points for the submission as indicated in the tasks. For each subtask you are given full marks for correct answers in your submission, 70% for minor mistakes, and 35%, 15%, or 0% for major mistakes or omissions depending on severity.

To mitigate against any form of plagiarism, including but not limited to the use of generative AI tools, **you MUST also present and explain your solution and confidently answer questions** about your submission to be graded. This oral assessment will take place during the scheduled labs after the submission deadline (or the week after, should we run out of time during these sessions).
If you do not show up your submission will not be graded unless there are approved IECs!

Your ability to fully explain your own submission will be crucial. You will only receive full marks if you are able to confidently present and explain all your work. Your assignment grade will be capped at 70% or 50% in case of minor issues with your explanations, and at 35%, 15%, or 0% in case of major problems with your explanations during the oral assessment.

The oral assessment will cover everything you uploaded onto Canvas, so be very careful what you include into your submission and make sure that you **only submit what you fully understand** to avoid your final marks being capped; in particular, do NOT include anything, including snippets taken from labs or lectures, unless you are able to explain it in detail or risk the whole rest of the assignment being marked down.

TASK 1 (adjacency matrix, 8 points)

In this task you will implement a graph data structure using adjacency matrices. Download the Java source files Assignment_NLDSA_1.zip from Canvas and edit the file *GraphAdjMatrix.java* to implement the missing code by replacing the `RuntimeException("Not yet implemented!")` where indicated.

A. Complete the constructor [0.5 points]

```
public GraphAdjMatrix(int noOfVertices, boolean directed)
```

   that given the number of vertices in the graph and a Boolean indicator if the graph should be directed or undirected initialises the data structure. Use `Double.NaN` in the adjacency matrix to indicate if two vertices are connected or not.

B. Complete the method [1 point]

```
public void addEdge(int source, int destination, double weight)
```

   that given a source vertex number, a destination vertex number, and the edge weight adds an additional edge to the graph. Make sure that your implementation works for directed and undirected graphs as initialised in the constructor.

C. Complete the method [1 point]

```
public void removeEdge(int source, int destination)
```

   that given a source vertex number and a destination vertex number removes this edge from the graph. Make sure that your implementation works for directed and undirected graphs as initialised in the constructor.

D. Complete the method [0.5 points]

```
public double getWeight(int source, int destination)
```

   that given a source vertex number and a destination vertex number returns the weight of the edge if it exists or `Double.NaN` otherwise.

E. Complete the method [1 point]

```
public int[] getNeighbours(int vertex)
```

   that given a vertex number returns the numbers of all adjacent vertices.

F. Complete the method [2 points]

```
public int getDegree(int vertex)
```

   that for a given vertex number returns its degree. Make sure that your implementation works for directed and undirected graphs as initialised in the constructor.

G. Complete the method [1 point]

```
public boolean isPath(int[] nodes)
```

that for a given list of vertex numbers determines if the sequence of nodes is on a path or not.

H. Complete the method [1 point]

```
public int getNoOfEdges()
```

that returns the number of edges of the graph. Your implementation must be based on the adjacency matrix (i.e. you should not count the number of edges in the add/remove methods) and must work for both directed and undirected graphs as initialised in the constructor.

TASK 2 (adjacency lists, 8 points)

In this task you will implement a graph data structure using adjacency lists. Download the Java source files Assignment_NLDSA_1.zip from Canvas and edit the file *GraphAdjList.java* to implement the missing code by replacing the `RuntimeException("Not yet implemented!")` where indicated.

A. Complete the constructor [0.5 points]

```
public GraphAdjList(int noOfVertices, boolean directed)
```

that given the number of vertices in the graph and a Boolean indicator if the graph should be directed or undirected initialises the data structure.

B. Complete the method [1 point]

```
public void addEdge(int source, int destination, double weight)
```

that given a source vertex number, a destination vertex number, and the edge weight adds an additional edge to the graph. Make sure that your implementation works for directed and undirected graphs as initialised in the constructor.

C. Complete the method [1 point]

```
public void removeEdge(int source, int destination)
```

that given a source vertex number and a destination vertex number removes this edge from the graph. Make sure that your implementation works for directed and undirected graphs as initialised in the constructor.

D. Complete the method [0.5 points]

```
public double getWeight(int source, int destination)
```

that given a source vertex number and a destination vertex number returns the weight of the edge if it exists or `Double.NaN` otherwise.

E. Complete the method [1 point]

```
public int[] getNeighbours(int vertex)
```

that given a vertex number returns the numbers of all adjacent vertices.

F. Complete the method [2 points]

```
public int getDegree(int vertex)
```

that for a given vertex number returns its degree. Make sure that your implementation works for directed and undirected graphs as initialised in the constructor.

G. Complete the method [1 point]

```
public boolean isPath(int[] nodes)
```

that for a given list of vertex numbers determines if the sequence of nodes is on a path or not.

H. Complete the method [1 point]

```
public int getNoOfEdges()
```

that returns the number of edges of the graph. Your implementation must be based on the adjacency lists (i.e. you should not count the number of edges in the add/remove methods) and must work for both directed and undirected graphs as initialised in the constructor.

TASK 3 (binary search trees, 9 points)

In this task you will implement a binary search tree data structure. Download the Java source files Assignment_NLDSA_1.zip from Canvas and edit the file *BinarySearchTreeImpl.java* to implement the missing code by replacing the `RuntimeException("Not yet implemented!")` where indicated.

A. Complete the method [2 points]

```
protected void insert(Node<T> x, int key, T value)
```

that given a node inside the tree inserts the given key-value pair into the subtree rooted at that node. Make sure to create a new node in case the key is not already stored in the subtree or replace the value for the key in the data structure otherwise.

B. Complete the method [1 point]

```
protected LinkedList<T> inorderTreeWalk(Node<T> x)
```

that given a node inside the tree returns the values of all nodes contained in the subtree rooted at that node.

C. Complete the method [0.5 points]

```
protected Node<T> search(Node<T> x, int key)
```

that given a node inside the tree returns the node containing the given key if this exists in the subtree rooted at the given node or `null` otherwise.

D. Complete the method [0.5 points]

```
protected int depth(Node<T> x)
```

that given a node inside the tree returns the depth of the subtree rooted at that node.

E. Complete the method [0.5 points]

```
protected Node<T> minimum(Node<T> x)
```

that given a node inside the tree returns the node containing the smallest key in the subtree rooted at the given node.

F. Complete the method [0.5 points]

```
protected Node<T> maximum(Node<T> x)
```

that given a node inside the tree returns the node containing the largest key in the subtree rooted at the given node.

G. Complete the method [1 point]

```
protected Node<T> successor(Node<T> x)
```

that given a node inside the tree returns the node containing the next larger key in the data structure.

H. Complete the method [1 point]

```
protected Node<T> predecessor(Node<T> x)
```

that given a node inside the tree returns the node containing the next smaller key in the data structure.

I. Complete the method [2 points]

```
protected void delete(Node<T> z)
```

that given a node inside the tree removes it from the data structure.