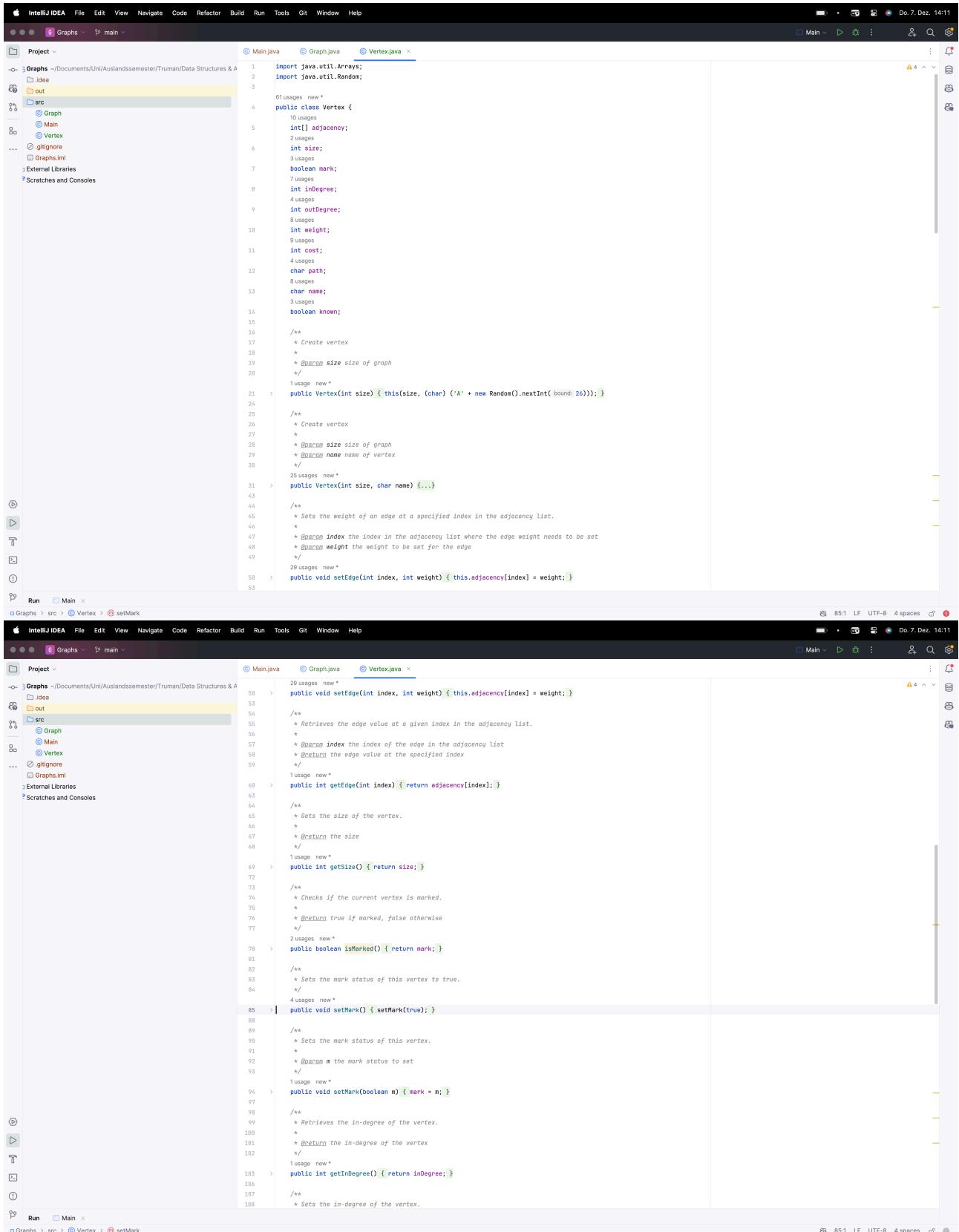


Bonus Assignment: Working with Graph

Thursday 7. December 2023

Deliverable 1

I implemented the vertex class as described in the slides.



The image shows two identical instances of the IntelliJ IDEA code editor side-by-side, displaying the same Java code for the `Vertex` class. Both instances have the same file path: `Graphs/src/Vertex.java`. The code is as follows:

```
import java.util.Arrays;
import java.util.Random;

public class Vertex {
    int[] adjacency;
    int size;
    boolean mark;
    int indegree;
    int outdegree;
    int weight;
    int cost;
    char path;
    String name;
    boolean known;

    /**
     * Create vertex
     *
     * @param size size of graph
     */
    public Vertex(int size) { this.size = (char) ('A' + new Random().nextInt(size)); }

    /**
     * Create vertex
     *
     * @param size size of graph
     * @param name name of vertex
     */
    public Vertex(int size, char name) {}

    /**
     * Sets the weight of an edge at a specified index in the adjacency list.
     *
     * @param index the index in the adjacency list where the edge weight needs to be set
     * @param weight the weight to be set for the edge
     */
    public void setEdge(int index, int weight) { this.adjacency[index] = weight; }

    /**
     * Retrieves the edge value at a given index in the adjacency list.
     *
     * @param index the index of the edge in the adjacency list
     * @return the edge value at the specified index
     */
    public int getEdge(int index) { return adjacency[index]; }

    /**
     * Gets the size of the vertex.
     *
     * @return the size
     */
    public int getSize() { return size; }

    /**
     * Checks if the current vertex is marked.
     *
     * @return true if marked, false otherwise
     */
    public boolean isMarked() { return mark; }

    /**
     * Sets the mark status of this vertex to true.
     */
    public void setMark() { mark = true; }

    /**
     * Sets the mark status of this vertex.
     *
     * @param m the mark status to set
     */
    public void setMark(boolean m) { mark = m; }

    /**
     * Retrieves the in-degree of the vertex.
     *
     * @return the in-degree of the vertex
     */
    public int getInDegree() { return indegree; }

    /**
     * Sets the in-degree of the vertex.
     */
    public void setInDegree(int n) { indegree = n; }
}
```

```

 186 /**
 187 * Sets the in-degree of the vertex.
 188 *
 189 * @param inDegree the in-degree value to set
 190 */
 191 public void setInDegree(int inDegree) { this.inDegree = inDegree; }

 192 /**
 193 * Retrieves the out-degree of the vertex.
 194 *
 195 * @return the out-degree of the vertex
 196 */
 197 public int getOutDegree() { return outDegree; }

 198 /**
 199 * Sets the out-degree of the vertex.
 200 *
 201 * @param outDegree the out-degree value to set
 202 */
 203 public void setOutDegree(int outDegree) { this.outDegree = outDegree; }

 204 /**
 205 * Calculates the total degree of the vertex by summing its in-degree and out-degree.
 206 *
 207 * @return the total degree of the node
 208 */
 209 public int getDegree() { return inDegree + outDegree; }

 210 /**
 211 * Retrieves the weight associated of the vertex.
 212 *
 213 * @return the weight
 214 */
 215 public int getWeight() { return weight; }

 216 /**
 217 * Sets the weight associated with the vertex.
 218 *
 219 * @param weight the weight value to set
 220 */
 221 public void setWeight(int weight) { this.weight = weight; }

 222 /**
 223 * Provides a string representation of the object.
 224 *
 225 * This includes the weight of the vertex and a string representation of its adjacency list.
 226 */
 227 @Override
 228 public String toString() {
 229     return weight + Arrays.toString(adjacency);
 230 }

```

I then created an Graph class which sorted my graph and all the sorting algorithms. It has different constructors. Some I needed for testing, others give the possibility to create own graphs and not imitate everything randomly.

IntelliJ IDEA 2023.3.1

File Edit View Navigate Code Refactor Build Run Tools Git Window Help

Graphs ~/Documents/Uni/Auslandssemester/Truman/Data Structures & A

Project

src

Main.java Graph.java Vertex.java

```

import java.util.*;
public class Graph {
    public Graph() {
        this(new Random().nextInt(10, 5000));
    }
    public Graph(Vertex[] graph) {
        this.size = graph.length;
        this.spareness = 0.5;
    }
    public Graph(int size) {
        this(size, spareness: 0.5);
    }
    public Graph(int size, double spareness) {
        Random random = new Random(seed: 3);
        int N = size;
        Vertex[] graph = new Vertex[N];
        for (int i = 0; i < N; i++) {
            graph[i] = new Vertex(N);
        }
        for (int i = 0; i < N; i++) {
            Vertex v = graph[i];
            for (int k = i + 1; k < N; k++) {
                if (random.nextDouble() < spareness) {
                    v.addEdge(k, weight);
                    v.setOutDegree(v.getOutDegree() + 1);
                    Vertex connectedVertex = graph[k];
                    connectedVertex.setInDegree(connectedVertex.getInDegree() + 1);
                }
            }
            v.weight = random.nextInt(bound: 100);
        }
        this.graph = graph;
    }
}

```

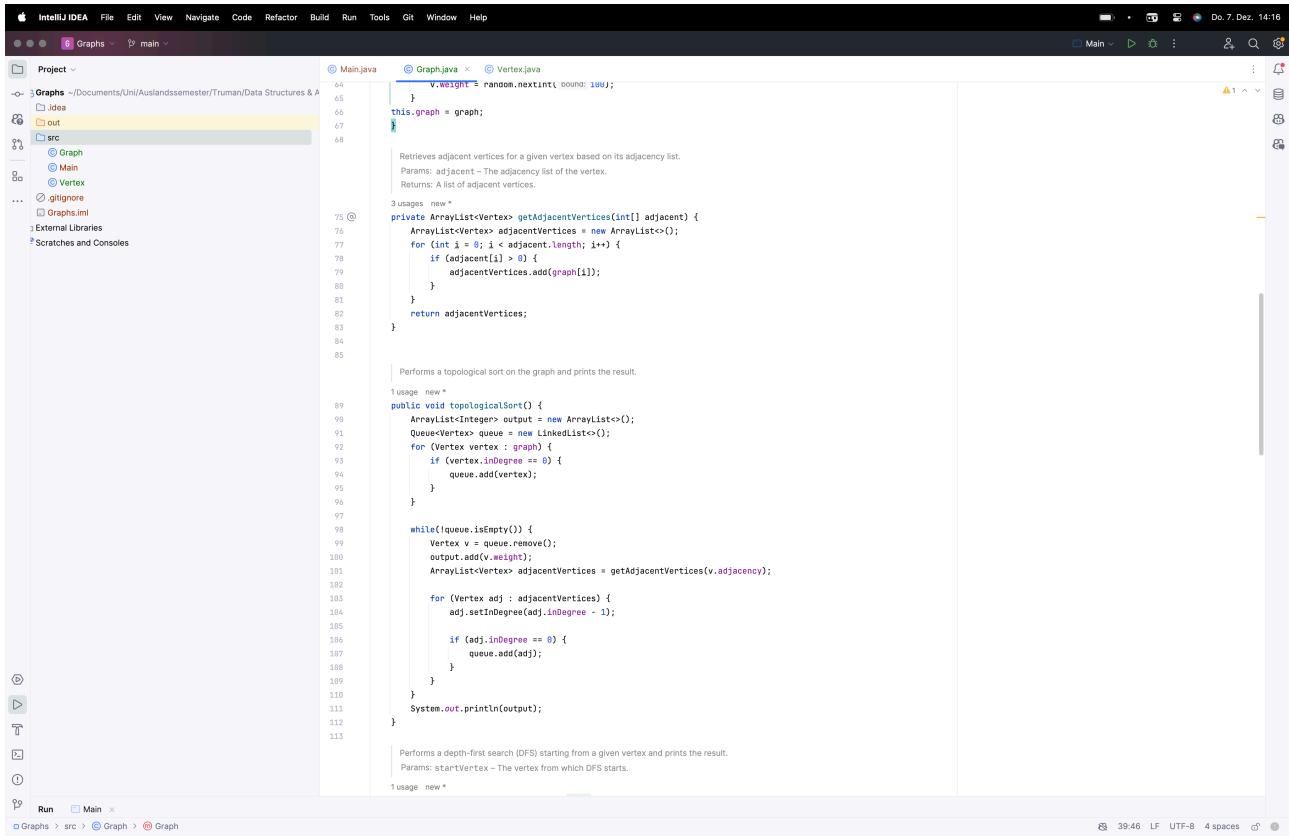
Run Main

Graphs > src > Graph > Graph

39:46 LF UTF-8 4 spaces

Deliverable 2

I implemented the sort algorithm based on the pseudo code in the slides



The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The project is named "Graphs" and contains a "src" directory with "Graph", "Main", and "Vertex" packages.
- Main.java:** This file contains the implementation of a topological sort algorithm. It includes imports for `java.util.*` and `Graph`.
- Graph.java:** This file contains the definition of the `Graph` class, which has a method `getAdjacentVertices` that returns an array of adjacent vertices.
- Vertex.java:** This file contains the definition of the `Vertex` class, which has an `inDegree` attribute and a `setInDegree` method.
- Code Editor:** The main editor shows the `Main.java` code, which uses the `getAdjacentVertices` method from the `Graph` class.
- Toolbars and Status Bar:** The top bar shows the IntelliJ IDEA logo, menu items like File, Edit, View, Navigate, Code, Refactor, Build, Run, Tools, Git, Window, Help, and a status bar indicating "Do. 7. Dez. 14:16". The bottom status bar shows "39:46 LF UTF-8 4 spaces ⌂".

I used the function `getAdjacentVertices` to get all adjacent vertices.

Deliverable 3:

These algorithms are also based on the pseudo code in the slides. They are very similar, the main difference is the data structure it uses to store the data for the next iteration.

The screenshot shows two side-by-side code editors in IntelliJ IDEA. The left editor contains Main.java and the right editor contains Graph.java. Both files are part of a project named 'Graphs' located at '/Documents/Uni/Auslandssemester/Truman/Data Structures & A'.

Main.java (Left Editor)

```
1 package Graphs;
2
3 public class Main {
4     public static void main(String[] args) {
5         Graph graph = new Graph();
6         graph.addEdge("A", "B");
7         graph.addEdge("A", "C");
8         graph.addEdge("B", "D");
9         graph.addEdge("C", "D");
10        graph.addEdge("D", "E");
11
12        System.out.println(graph.getAdjacentVertices("A"));
13    }
14 }
```

Graph.java (Right Editor)

```
1 package Graphs;
2
3 import java.util.ArrayList;
4 import java.util.LinkedList;
5 import java.util.Queue;
6
7 public class Graph {
8     private ArrayList<Vertex> vertices;
9
10    public Graph() {
11        vertices = new ArrayList<Vertex>();
12    }
13
14    public void addEdge(String source, String target) {
15        Vertex sourceVertex = getVertex(source);
16        Vertex targetVertex = getVertex(target);
17
18        if (sourceVertex == null || targetVertex == null) {
19            return;
20        }
21
22        sourceVertex.adjacency.add(targetVertex);
23        targetVertex.adjacency.add(sourceVertex);
24    }
25
26    public ArrayList<Vertex> getAdjacentVertices(String vertexName) {
27        Vertex vertex = getVertex(vertexName);
28
29        if (vertex == null) {
30            return null;
31        }
32
33        return vertex.adjacency;
34    }
35
36    public void printGraph() {
37        for (Vertex vertex : vertices) {
38            System.out.println(vertex.name + ": " + vertex.adjacency);
39        }
40    }
41
42    private Vertex getVertex(String name) {
43        for (Vertex vertex : vertices) {
44            if (vertex.name.equals(name)) {
45                return vertex;
46            }
47        }
48
49        return null;
50    }
51 }
```

Vertex.java (Bottom Editor)

```
1 package Graphs;
2
3 public class Vertex {
4     private String name;
5     private ArrayList<Vertex> adjacency;
6
7     public Vertex(String name) {
8         this.name = name;
9         adjacency = new ArrayList<Vertex>();
10    }
11
12    public void setInDegree(int inDegree) {
13        this.inDegree = inDegree;
14    }
15
16    public int getInDegree() {
17        return inDegree;
18    }
19
20    public void add(Vertex adj) {
21        adjacency.add(adj);
22    }
23
24    public ArrayList<Vertex> getAdjacency() {
25        return adjacency;
26    }
27
28    private int inDegree;
29 }
```

Deliverable 4

I implemented the algorithm based on the Pseudo code. Then I implemented helper functions called printShortestPaths, printPath and findVertexByName which I call after sorting to print out all the shortest paths. The printing statements run into a bit of a problem if there are vertices with the same name. Because I only use uppercase letters (36), there can easily occur an duplication of names. I didn't fix it, because in the example it works, and I think it is safe to assume, that the names will be different.

```
graph TD; subgraph Main_file [Main.java]; 162; 163; 164; 165; 166; 167; 168; 169; 170; 171; 172; 173; 174; 175; 176; 177; 178; 179; 180; 181; 182; 183; 184; 185; 186; 187; 188; 189; 190; 191; 192; 193; 194; 195; 196; 197; 198; 199; 200; 201; 202; 203; 204; 205; 206; 207; 208; 209; 210; 211; 212; 213; 214; 215; 216; 217; 218; 219; 220; 221; 222; 223; 224; 225; 226; 227; 228; 229; 230; 231; 232; 233; 234; 235; 236; 237; 238; 239; 240; 241; 242; 243; 244; 245; 246; 247; 248; 249; 250; 251; 252; 253; 254; 255; 256; 257;
```

```
graph TD; subgraph Graph_file [Graph.java]; 162; 163; 164; 165; 166; 167; 168; 169; 170; 171; 172; 173; 174; 175; 176; 177; 178; 179; 180; 181; 182; 183; 184; 185; 186; 187; 188; 189; 190; 191; 192; 193; 194; 195; 196; 197; 198; 199; 200; 201; 202; 203; 204; 205; 206; 207; 208; 209; 210; 211; 212; 213; 214; 215; 216; 217; 218; 219; 220; 221; 222; 223; 224; 225; 226; 227; 228; 229; 230; 231; 232; 233; 234; 235; 236; 237; 238; 239; 240; 241; 242; 243; 244; 245; 246; 247; 248; 249; 250; 251; 252; 253; 254; 255; 256; 257;
```

Phillip Lechenauer