Unit Testing Plan

1.0 List of the Methods Under Test

1.1 Dijkstra.java

- run(Node startingNode): Map<String[], String[]>
- getComparisons(): int

1.2 Node.java

- setName(String name): void
- getLabel(): Text
- getShape(): Circle
- setActive(): void
- setInActive(): void
- compareTo(Node other): int
- equals(Object other): boolean
- toString(): String

1.3 Edge.java

- getId(): int
- getNode1(): Node
- getNode2(): Node
- setWeight(String weight): void
- getLabel(): Text
- getShape(): Line
- setActive(): void
- setInactive(): void
- toString(): String

1.4 ComparisonChartData.java

- generateGraph(int numberOfNodes): void
- getComparisons(int numberOfNodes): int

1.5 Database.java

- getNodes(): List<Circle>
- setStartNode(Circle nodeShape): void
- setActive(Circle nodeShape): void
- setActive(Line edgeShape): void

- setInactive(Circle nodeShape): void
- setInactive(Line edgeShape): void
- getAdjacentNodesAndEdges(Circle nodeShape): Map<Circle, Line>
- addNode(Text label, Circle node): void
- addEdge(Circle nodeShape1, Circle nodeShape2, Text label, Line edge): void
- removeNode(Circle nodeShape): void
- removeEdge(Line edgeShape): void
- findLabel(Circle nodeShape): Text
- findLabel(Line edgeShape): Text
- getNodeName(int nodeId): String
- getNodeShape(int nodeId): Circle
- getAttachedEdges(Circle nodeShape): List<Line>
- edgeExists(Circle nodeShape1, Circle nodeShape2): boolean
- updateLabel(Text label, String newText): void
- runDijkstra(): Map<String[], String[]>
- clear(): void
- getNodesAndLabels(): Map<Circle, Text>
- getEdgesAndLabels(): Map<Line, Text>

2.0 Test Techniques

I will use black box testing techniques to identify potential edge cases and validate input handling. To ensure the correctness and robustness of the system, I will use equivalence class partitioning and boundary value analysis.

3.0 Designing the Unit Test-Cases

3.1 Dijkstra.java

run(Node startingNode): Map<String[], String[]>

Equivalence class partitioning:

a. Valid starting node: A node that exists in the graph.
b. Invalid starting node: A node that does not exist in the graph.

Boundary value analysis:

a. Starting node: A node with one edge.

b. Starting node: A node with multiple edges.

Test cases:

1. Test with a node with one edge.
2. Test with a node with 3 edges.
3. Test with an invalid starting node.
4. Test with a null starting node.

getComparisons(): int

Equivalence class partitioning and boundary value analysis do not directly apply to this method, but these test cases ensure its correctness:

1. Test with a linear graph with a small number of comparisons.
2. Test with a complete graph with a large number of comparisons.

3.2 Node.java

setName(String name): void

Equivalence class partitioning:

a. Valid name: A non-empty string.

b. Invalid name: An empty string or a null value.

Boundary value analysis:

a. Name: A single character.

b. Name: A long string.

Test cases:

1. Test with a single character.
2. Test with a long string.
3. Test with an empty string.
4. Test with a null name.

compareTo(Node other): int

Equivalence class partitioning:

a. Valid node: A non-null node.

b. Invalid node: A null value.

Boundary value analysis:

a. Node with a name that comes first alphabetically.

b. Node with a name that comes last alphabetically.

Test cases:

1. Test with a node with a name that comes first alphabetically.
2. Test with a node with a name that come last alphabetically.
3. Test with a null node.

equals(Object other): boolean

Equivalence class partitioning:

a. Valid object: A non-null node object.
b. Invalid object: A null value or an object of a different class.

Boundary value analysis:

a. Node with the same name.
b. Node with a different name.

Test cases:

1. Test with a node with the same name.
2. Test with a node with a different name.
3. Test with a null value.
4. Test with a string object.

getLabel(): Text

getShape(): Circle

setActive(): void

setInActive(): void

toString(): String

Test these methods for correctness with valid data.

3.3 Edge.java

setWeight(String weight): void

Equivalence class partitioning:

a. Valid weight: A integer.
b. Invalid weight: A non-integer string or negative integer.

Boundary value analysis:

a. Zero.
b. A large integer.

Test cases:

1. Test with a positive integer.
2. Test with a large positive integer.
3. Test with a negative integer.
4. Test with a non-integer string.

getId(): int

getNode1(): Node

getNode2(): Node

getLabel(): Text

getShape(): Line

setActive(): void

setInactive(): void

toString(): String

Test these methods for correctness with valid data.

3.4 ComparisonChartData.java

generateGraph(int numberOfNodes): void

Equivalence class partitioning:

a. Valid number of nodes: A positive integer.
b. Invalid number of nodes: A negative integer.

Boundary value analysis:

a. Minimum number of nodes.
b. Invalid value just below the minimum number of nodes.

Test cases:

1. Test with two nodes.
2. Test with a negative integer.
3. Test with zero nodes.

getComparisons(int numberOfNodes): int

Equivalence class partitioning:

a. Valid number of nodes: A positive integer.
b. Invalid number of nodes: A negative integer.

Boundary value analysis:

a. Minimum number of nodes.

b. Invalid value just below the minimum number of nodes.

Test cases:

1. Test with two nodes.
2. Test with a negative integer.
3. Test with zero nodes.

3.5 Database.java

getAdjacentNodesAndEdges(Circle nodeShape): Map<Circle, Line>

Equivalence class partitioning:

a. Valid input: A non-null circle object.
b. Invalid input: A null value.

Boundary value analysis:

N/A

Test cases:

1. Test with a valid circle object.
2. Test with a null value.

addNode(Text label, Circle node): void

Equivalence class partitioning:

a. Valid input: Non-null circle and text objects.
b. Invalid input: Null values.

Boundary value analysis:

N/A

Test cases:

1. Test with valid circle and test objects.
2. Test with null values.

addEdge(Circle nodeShape1, Circle nodeShape2, Text label, Line edge): void

Equivalence class partitioning:

a. Valid input: Non-null circle, text and line objects.
b. Invalid input: Null values.

Boundary value analysis:

N/A

Test cases:

1. Test with valid circle, text and line objects.
2. Test with null values.

removeNode(Circle nodeShape): void

Equivalence class partitioning:

a. Valid input: A non-null circle object.
b. Invalid input: A null value.

Boundary value analysis:

N/A

Test cases:

1. Test with a valid circle object.
2. Test with a null value.

removeEdge(Line edgeShape): void

Equivalence class partitioning:

a. Valid input: A non-null line object.
b. Invalid input: A null value.

Boundary value analysis:

N/A

Test cases:

1. Test with a valid line object.
2. Test with a null value.

findLabel(Circle nodeShape): Text

Equivalence class partitioning:

a. Valid input: A non-null circle object.
b. Invalid input: A null value.

Boundary value analysis:

N/A

Test cases:

1. Test with a valid circle object.
2. Test with a null value.

findLabel(Line edgeShape): Text

Equivalence class partitioning:

a. Valid input: A non-null line object.

b. Invalid input: A null value.

Boundary value analysis:

N/A

Test cases:

1. Test with a valid line object.
2. Test with a null value.

getAttachedEdges(Circle nodeShape): List<Line>

Equivalence class partitioning:

a. Valid input: A non-null circle object.
b. Invalid input: A null value.

Boundary value analysis:

N/A

Test cases:

1. Test with a valid circle object.
2. Test with a null value.

edgeExists(Circle nodeShape1, Circle nodeShape2): boolean

Equivalence class partitioning:

a. Valid input: Non-null circle objects.
b. Invalid input: Null values.

Boundary value analysis:

N/A

Test cases:

1. Test with valid circle objects.
2. Test with null values.

updateLabel(Text label, String newText): void

Equivalence class partitioning:

a. Valid input: Non-null text and string objects.
b. Invalid input: Null values.

Boundary value analysis:

N/A

Test cases:

1. Test with valid text and string obejcts
2. Test with null values.

runDijkstra(): Map<String[], String[]>

Equivalence class partitioning:

a. Valid starting node: A node that exists in the graph.
b. Invalid starting node: A node that does not exist in the graph.

Boundary value analysis:

a. Starting node: A node with one edge.
b. Starting node: A node with multiple edges.

Test cases:

1. Test with a node with one edge.
2. Test with a node with 3 edges.
3. Test with an invalid starting node.
4. Test with a null starting node.

getNodes(): List<Circle>

setStartNode(Circle nodeShape): void

setActive(Circle nodeShape): void

setActive(Line edgeShape): void

setInactive(Circle nodeShape): void

setInactive(Line edgeShape): void

getNodeName(int nodeId): String

getNodeShape(int nodeId): Circle

clear(): void

getNodesAndLabels(): Map<Circle, Text>

getEdgesAndLabels(): Map<Line, Text>

Test these methods for correctness with valid data.