# **User Documentation for com.septim.graphlib Graph Library**

## **Overview**

The **com.septim.graphlib** package provides tools for creating, visualizing, and performing operations on graphs. This documentation covers the **Graph** and **Prims** classes, detailing their functionalities and usage.

## **Graph Class**

The **Graph** class represents a graph and provides methods to visualize it in a grid layout.

### **Properties**

* **N**: The number of vertices in the graph.
* **edges**: A 2D array representing the edges as pairs of vertices.
* **edgeSides**: An array specifying the side (left or right) on which the edge is displayed in the grid.
* **edge\_colors**: A map of edge indices to their respective colors using **Attribute** from **jcolor**.
* **vertex\_colors**: A map of vertex indices to their respective colors using **Attribute** from **jcolor**.

### **Methods**

#### **void edges(int... vertices)**

Sets the edges of the graph.

* **vertices**: A variable number of integers representing pairs of vertices.

#### **void setEdgeSides(Side[] edgeSides)**

Sets the sides for the edges.

* **edgeSides**: An array of **Side** enum values specifying the side for each edge.

#### **void set\_perms(int[] perm\_x, int[] perm\_y)**

Sets the permutations for the x and y axes.

* **perm\_x**: Permutation for the x axis.
* **perm\_y**: Permutation for the y axis.

#### **void print\_graph()**

Prints the graph in a grid layout, displaying edges and vertices according to their permutations and sides.

### **Enums**

#### **Side**

Specifies the side on which an edge is displayed in an L shape.

* **left**: The edge is displayed on the left.
* **right**: The edge is displayed on the right.

#### **Direction**

Represents the direction for grid connections.

* **top**
* **left**
* **right**
* **bottom**

### **Helper Classes**

#### **Cell**

An interface representing a cell in the grid.

##### **Methods**

* **static Cell set(Cell cell, char c, int edge)**: Sets the cell with the specified character and edge.
* **static boolean connects\_horizontally(Cell a, Cell b)**: Checks if two cells connect horizontally.
* **boolean connects\_from(Direction dir)**: Checks if the cell connects from a specific direction.

#### **Node**

Represents a vertex in the grid.

* **vertex**: The vertex number.

#### **Path**

Represents a path in the grid.

* **c**: The character representing the path.
* **edge\_from**: An array storing edges for each direction.

## **Prims Class**

The **Prims** class implements Prim's algorithm for finding the minimum spanning tree (MST) of a graph.

### **Properties**

* **callback**: A **Callback** interface instance to be called during the algorithm's execution.
* **graph**: The graph on which Prim's algorithm is run.
* **start**: The starting vertex for the algorithm.
* **weights**: An array of weights for the edges.

### **Methods**

#### **Prims()**

Default constructor. Initializes a sample graph with predefined edges, weights, and permutations.

#### **Prims(Graph graph, int start, int[] weights)**

Parameterized constructor. Initializes the graph, start vertex, and edge weights based on the provided parameters.

#### **List<Integer> run() throws IOException**

Runs Prim's algorithm on the graph and returns the MST. If no callback is set, a default callback is used to print the graph.

### **Interfaces**

#### **Callback**

Defines a method to be called during the algorithm's execution.

* **void call(int currVertex, int prevVertex) throws IOException**: Method to be implemented for callback functionality.

## **Dijkstra Class**

The **Dijkstra** class implements Dijkstra's algorithm for finding the shortest paths from a source vertex to all other vertices in a graph.

### **Properties**

* **graph**: The graph on which Dijkstra's algorithm is run.
* **source**: The source vertex for the algorithm.
* **distances**: An array of distances from the source vertex to each other vertex.
* **previous**: An array to store the previous vertex on the shortest path from the source.

### **Methods**

#### **Dijkstra(Graph graph, int source)**

Constructor. Initializes the graph and the source vertex for the algorithm.

* **graph**: The graph on which the algorithm will run.
* **source**: The source vertex from which shortest paths are calculated.

#### **void run()**

Runs Dijkstra's algorithm to calculate the shortest paths from the source vertex to all other vertices in the graph.

#### **List<Integer> getPath(int target)**

Retrieves the shortest path from the source vertex to the specified target vertex.

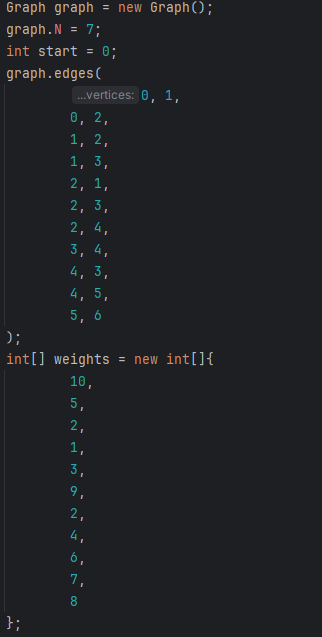
* **target**: The target vertex for which the shortest path is retrieved.
* **Returns**: A list of vertices representing the shortest path from the source to the target.

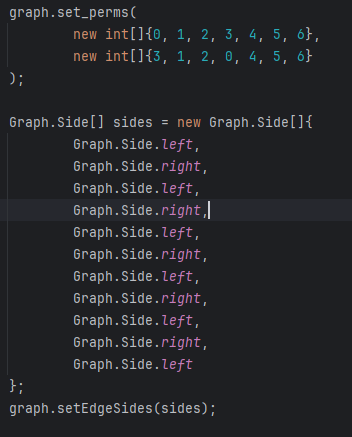
## **Dependencies**

* **jcolor**: Used for coloring vertices and edges in the graph visualization.

## **Notes**

* Ensure the number of vertices specified in **edges(int... vertices)** is even.
* The permutations **perm\_x** and **perm\_y** define the grid layout of the graph. Each vertex is displayed in its own row/column based on these permutations.





More examples can be seen in Main.java.