GTU Department of Computer Engineering

CSE 222 / 505 – Spring 2022

Homework 8 - Report

Ahmet USLUOGLU

# 1801042602

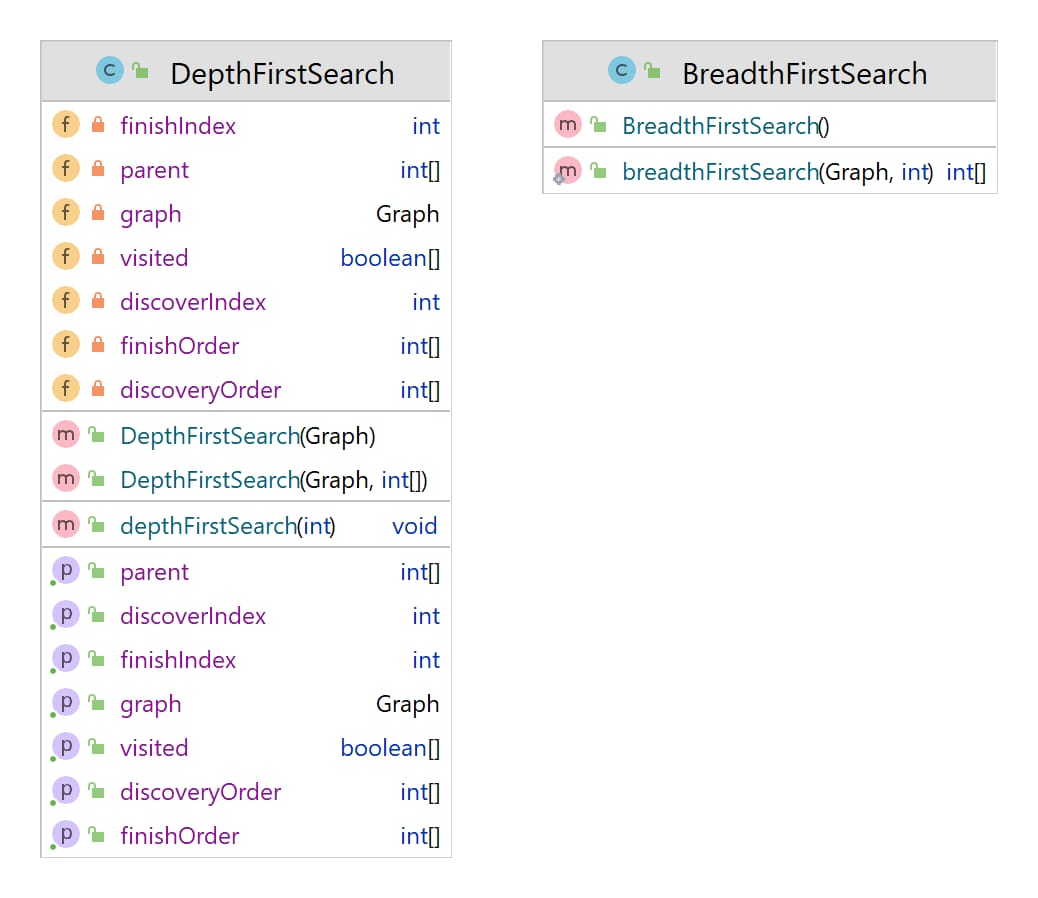
### – System Requirement

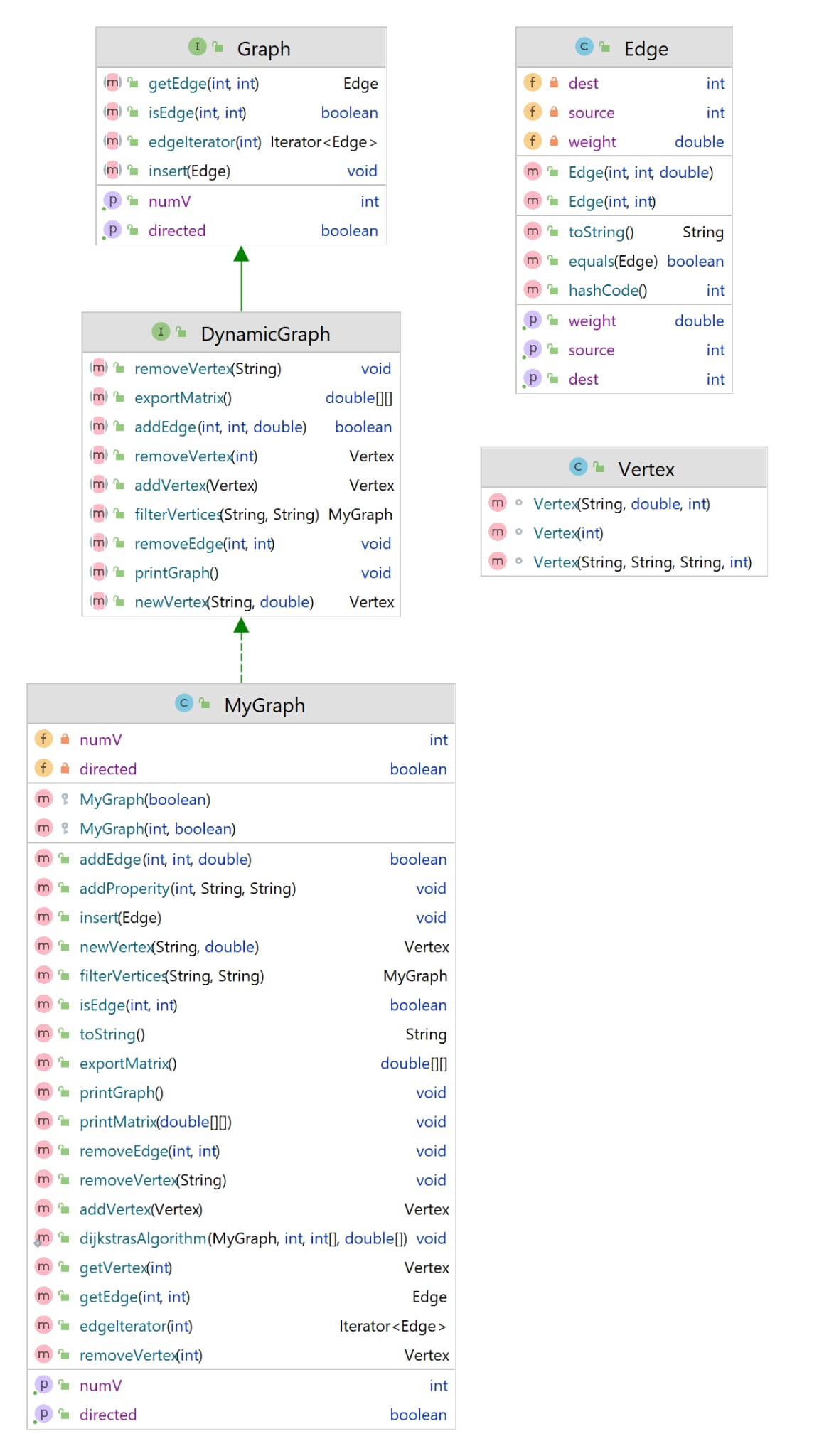
Operating System must have JDK (Java Development Kit) 11 and JRE (Java Runtime Environment) 11 or higher.

There should be enough space for storing data’s.

1. – Class Diagrams

\*Higher Resolution Version of the Class Diagram is in the files.





3. Problem - Solution Approach

Problem:

1 - ) Define a DynamicGraph interface by extending the Graph interface in the book for the following definition of graph data structure. Write a MyGraph class for the implementation of DynamicGraph interface. In your implementation, define a Vertex class for representing the vertices in the graph. A vertex must have an index (ID), a label, and a weight. The vertices may have user-defined additional properties (Vertex class should be generic), so you have to handle this requirement. Use adjacency list representation to handle the edges between vertices in the graph data structure.

2-) Write a method that takes a MyGraph object as a parameter and performs BFS and DFS traversals. The method calculates the total distance of the path for accessing each vertex during the traversal, and it returns the difference between the total distances of two traversal methods. If there are more than one alternative to access a vertex at a specific level during the BFS, the shortest alternative should be considered. The vertices should be considered in distance order during DFS traversal, so, from a vertex v, DFS should continue with a vertex w which has the smallest edge from v, among all adjacent vertices of v.

3-)Write a method that takes a MyGraph object and a vertex as a parameter to perform a modified version of Dijkstra’s Algorithm for calculating the shortest paths from the given vertex to all other vertices in the graph. In this modified version, the algorithm considers boosting value of the vertices in addition to the edge weights. The boosting property is a user defined property that takes double values. The boosting values are subtracted from the total length of paths that they are contained in

Solutions:

1-) I have used HashMap to represent Adjacency List in MyGraph class which implements DynamicGraph Interface. This way most of the implementations are Constant Time.

I have used ID as key and Vertex as Value of the HashMap.

Every Vertex keeps Edge List which the source of each edge is The Vertex itself.

2-) I have used BFS and DFS examples in the book and modified the source code to our needs.

Each iteration in the edge list follows the least weighted edge. After taking the array of visited vertexes in each search.

I have added the total distance between those edges.

I have subtracted the total distances that I get from DFS and BFS to calculate difference.

3-) I have modified the Dijkstra’s Algorithm by Subtracting Boosting Values of vertexes (if there is any) from the edge’s weights that we iterated through except the start and final destination vertexes.

Calculated the least weighted path by substracting boosting value.

1. – Test Cases

a-) Testing the part1

1 – Creating an empty graph with n vertices.

2 – Adding new 2 vertices and 1 edge.

3 –Creating an empty graph and inserting edges.

4- Adding a new vertex.

5- Adding a new edge between empty vertexes.

6- Deleting an edge.

7- Deleting a vertex.

8- Exporting matrix version and print.

9- Testing isEdge function.

10- Adding Filter to Vertexes and Creating a subgraph.

b-) Testing the part2

1 -) Testing the Depth first search distance

2 -) Testing the Breadth first search distance

3- Testing the Difference between DFS and BFS.

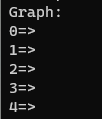
c-) Testing the part3

1 -) Testing the Dijkstra’s Algorithm

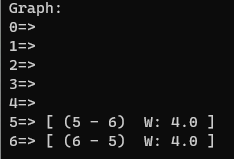
## 5 – Running Program and Results

a-) Testing the part1

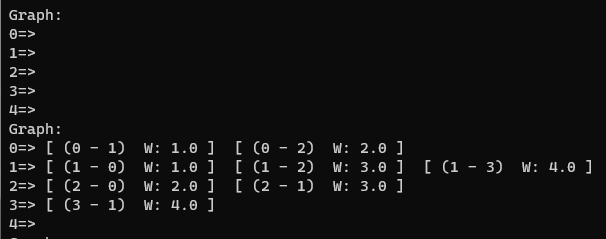
1 – Creating an empty graph with n vertices.



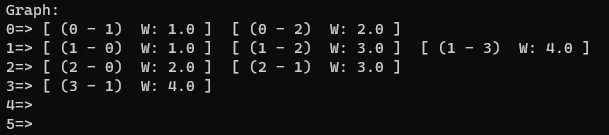
2 – Adding new 2 vertices and 1 edge.



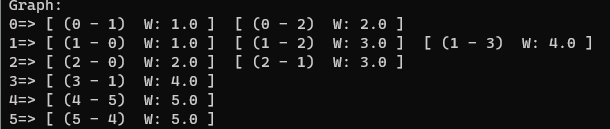
3 –Creating an empty graph and inserting edges.



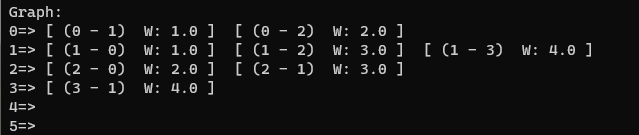
4- Adding a new vertex.



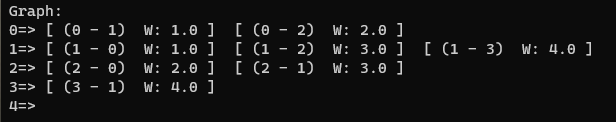
5- Adding a new edge between empty vertexes.



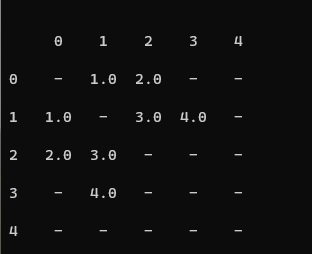
6- Deleting an edge.



7- Deleting a vertex.



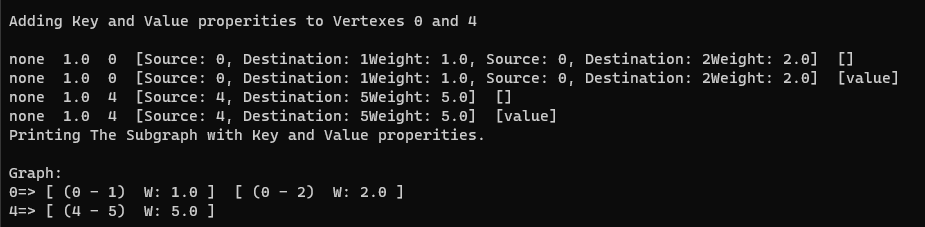
8- Exporting matrix version and print.



9- Testing isEdge function.

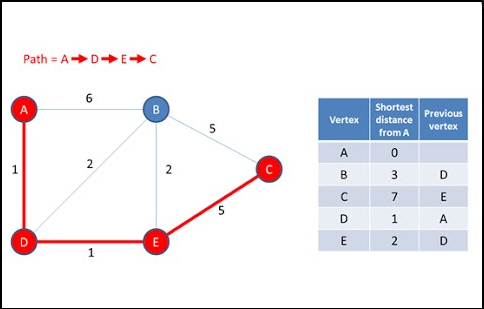


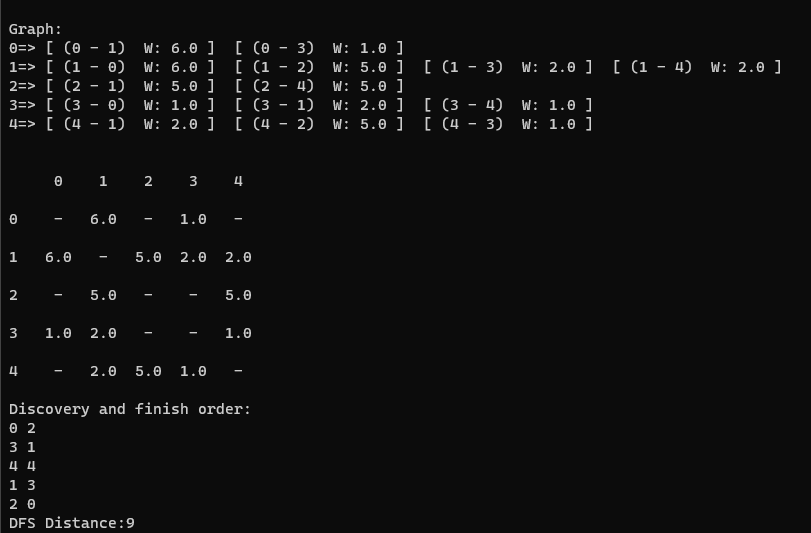
10- Adding Filter to Vertexes and Creating a subgraph.



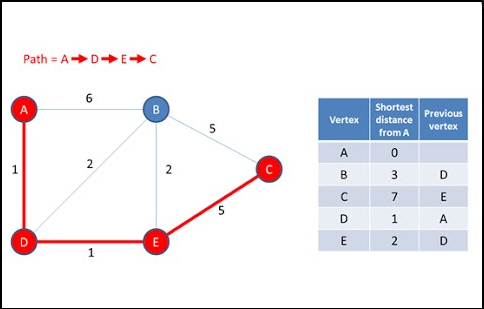
b-) Testing the part2

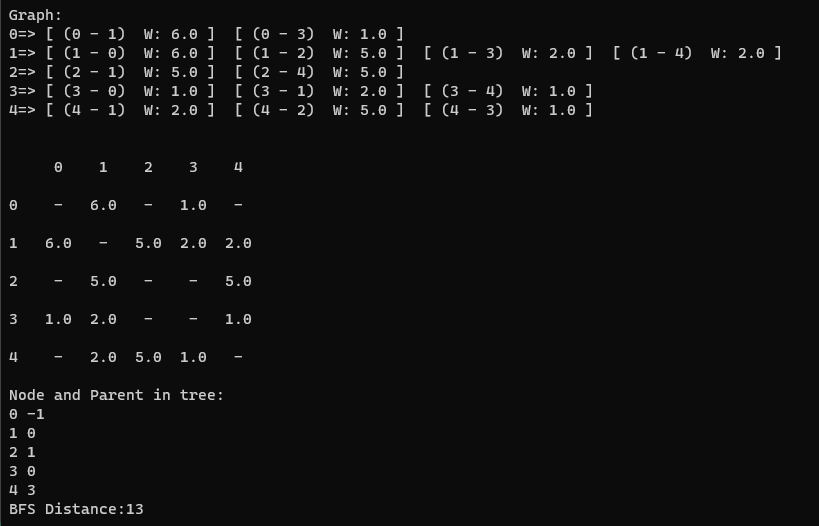
1 -) Testing the Depth first search distance



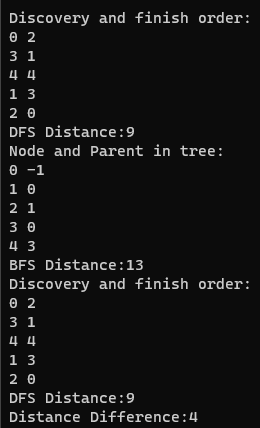


2 -) Testing the Breadth first search distance



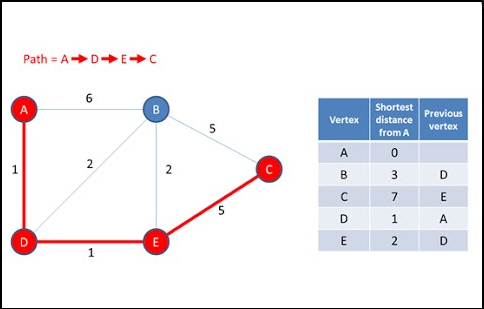


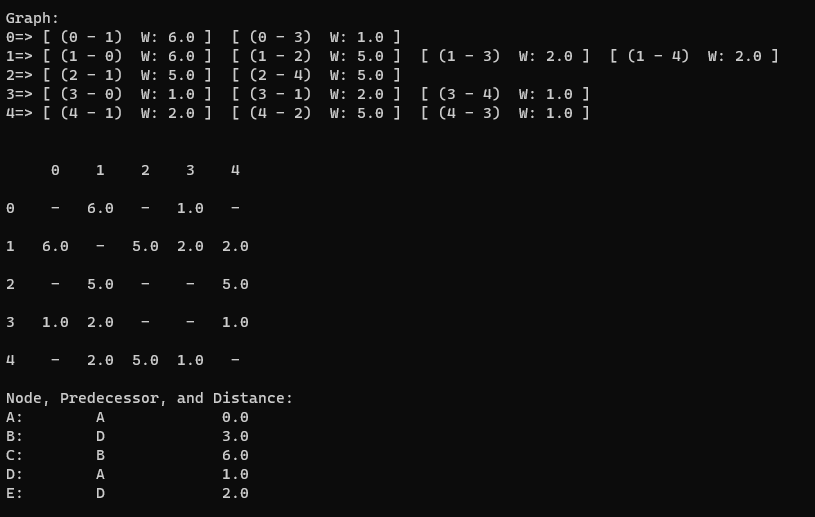
3- Testing the Difference between DFS and BFS.

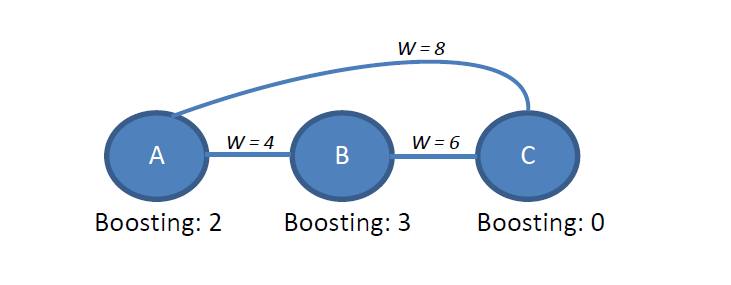


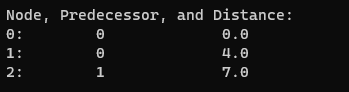
c-) Testing the part3

1 -) Testing the Dijkstra’s Algorithm





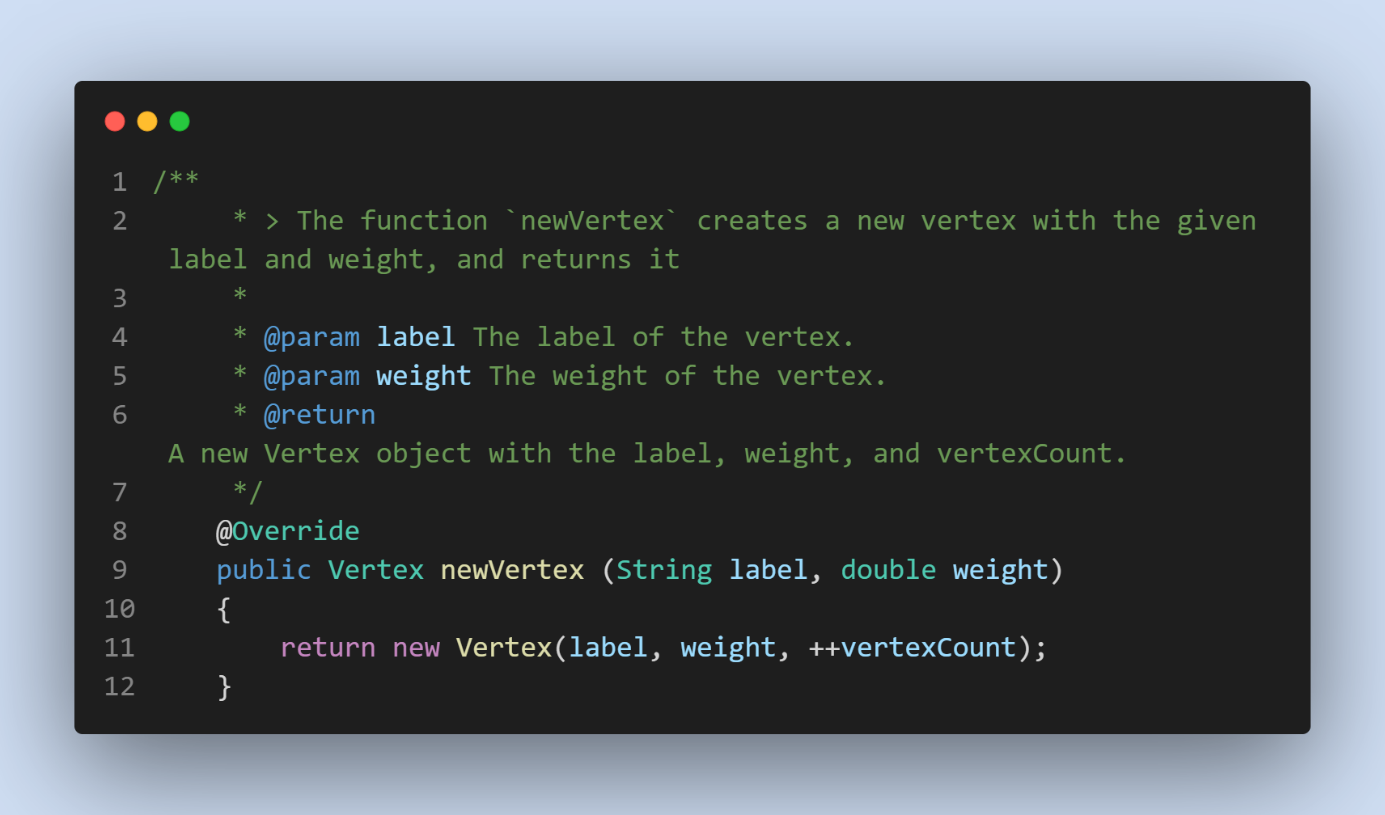




### 6 – Calculate Time complexity

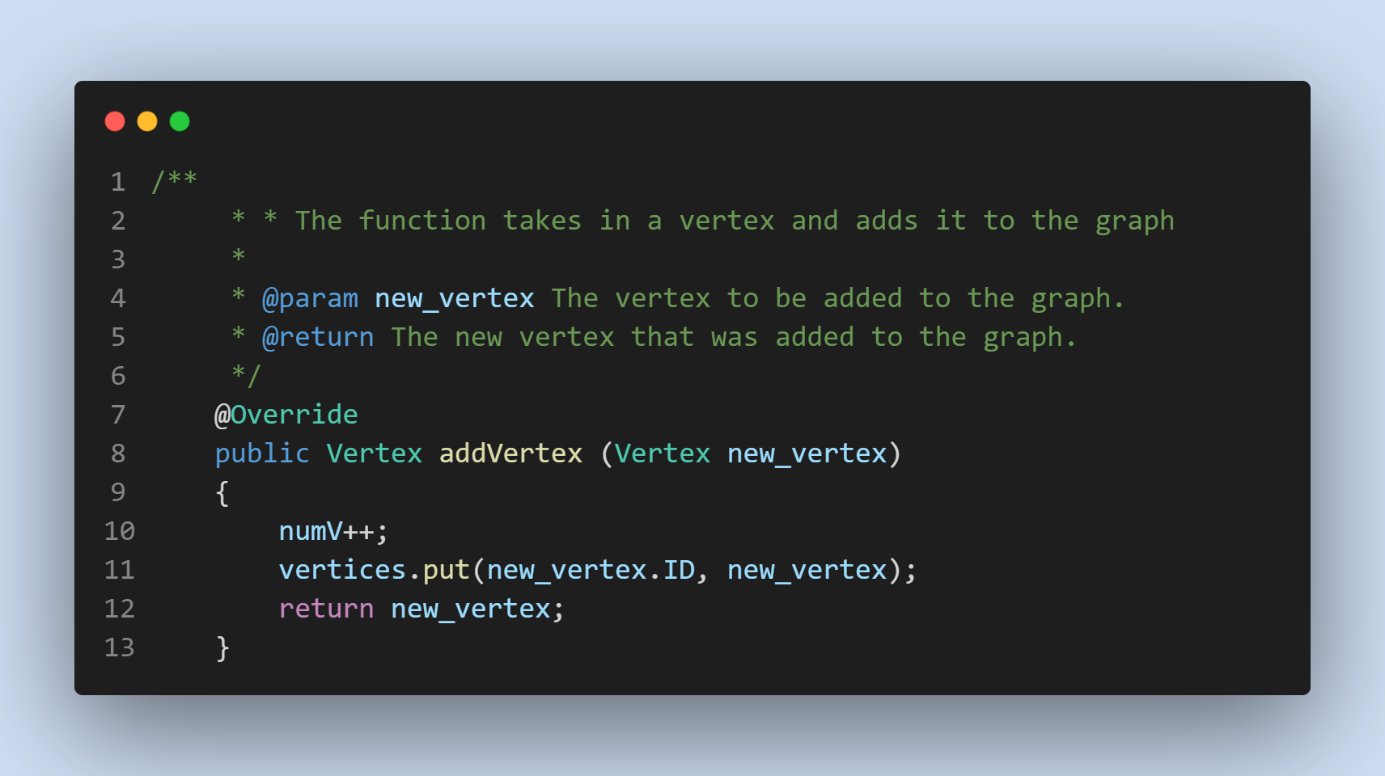
1- ) Vertex newVertex (String label, double weight);

Complexity O(1).



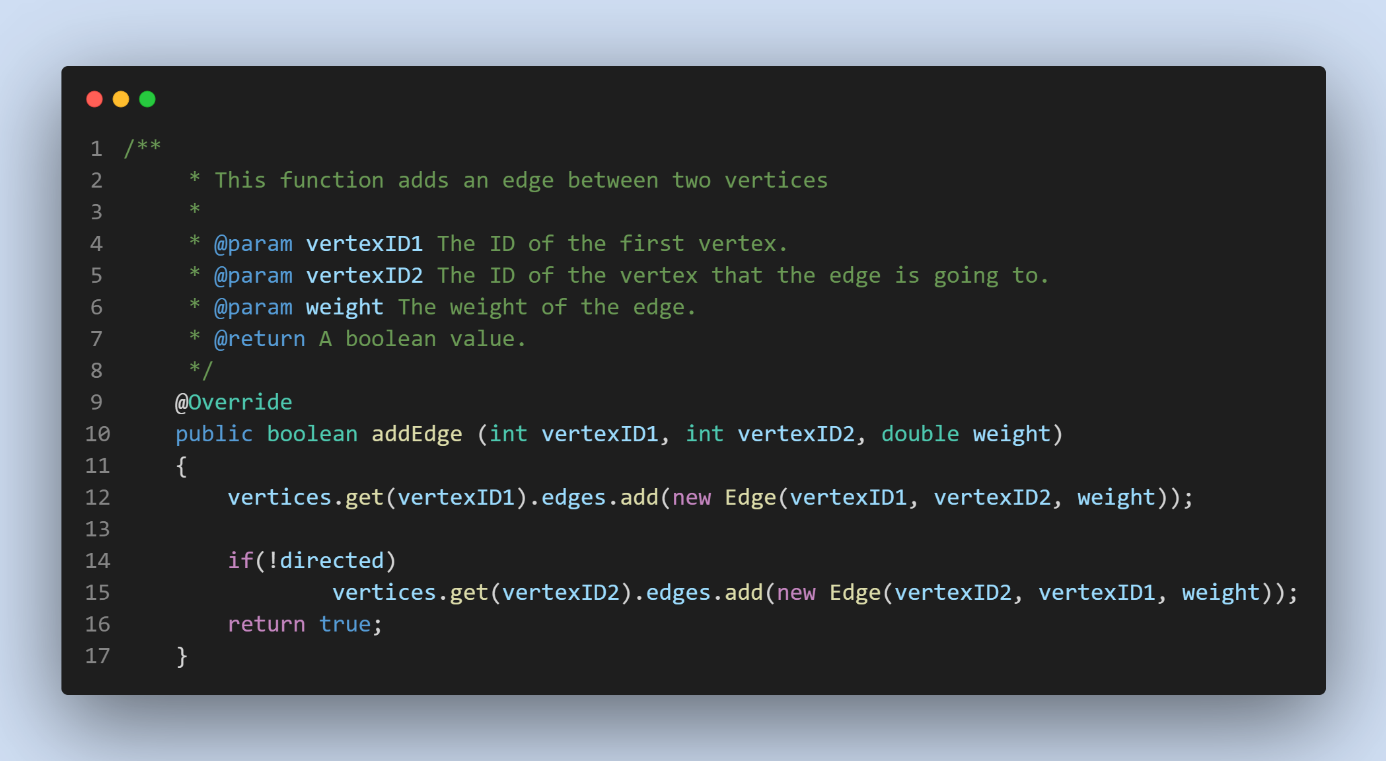
2-) Vertex addVertex (Vertex new\_vertex);

Complexity O(1).



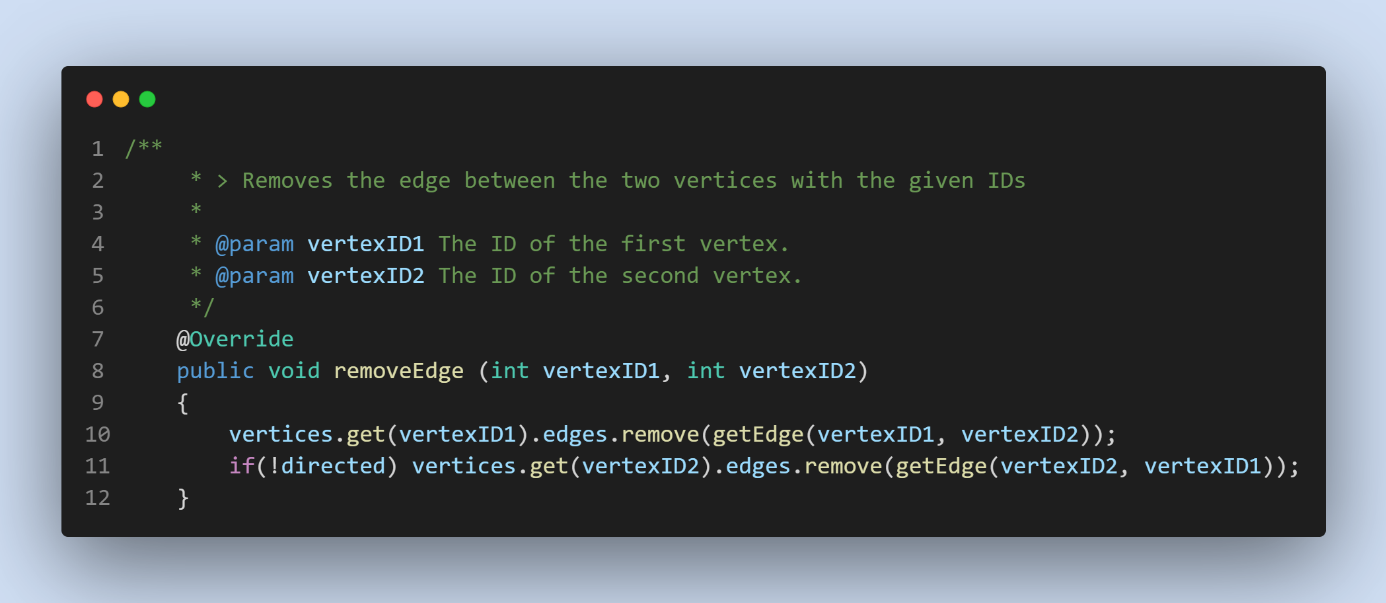
3-) boolean addEdge (int vertexID1, int vertexID2, double weight);

Complexity O(1).



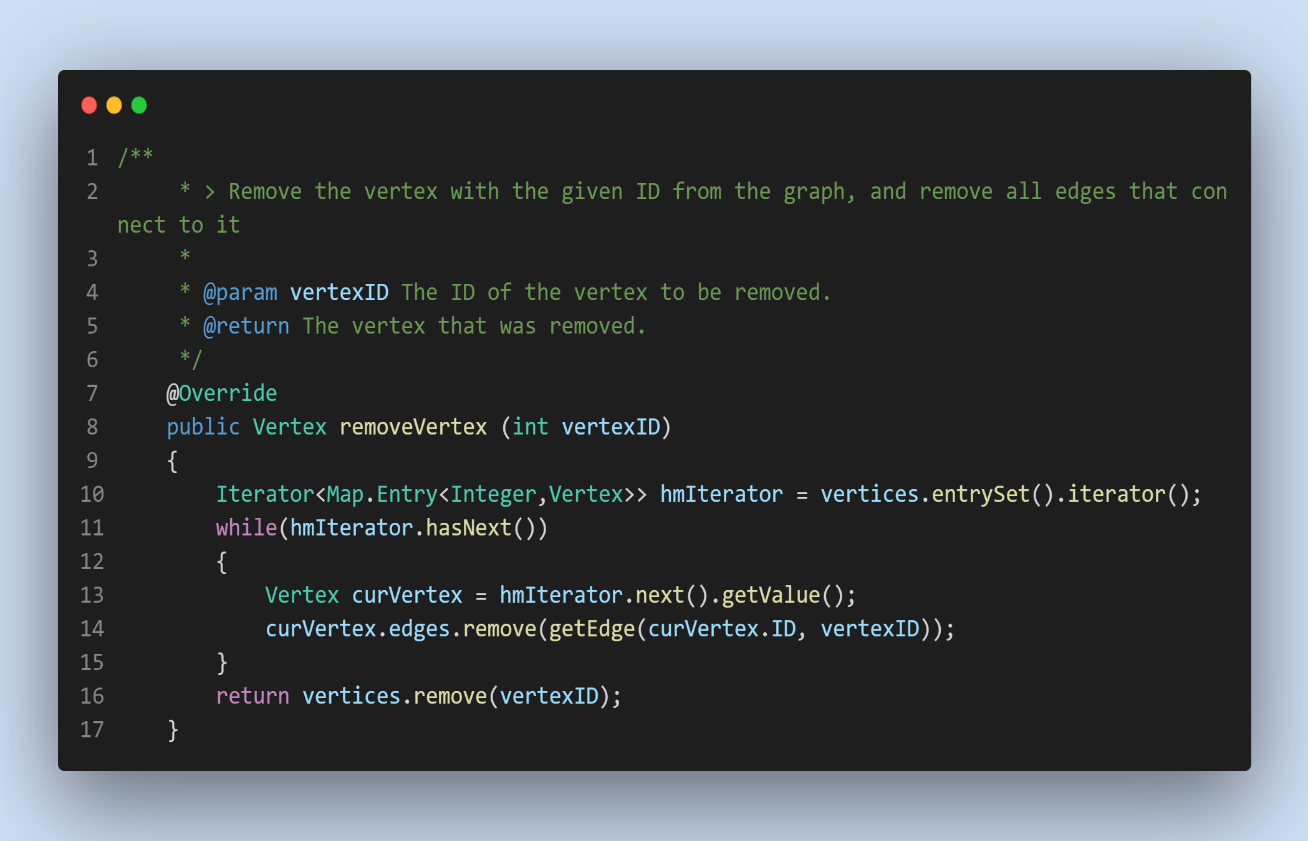
4 -) void removeEdge (int vertexID1, int vertexID2);

Complexity O(1).



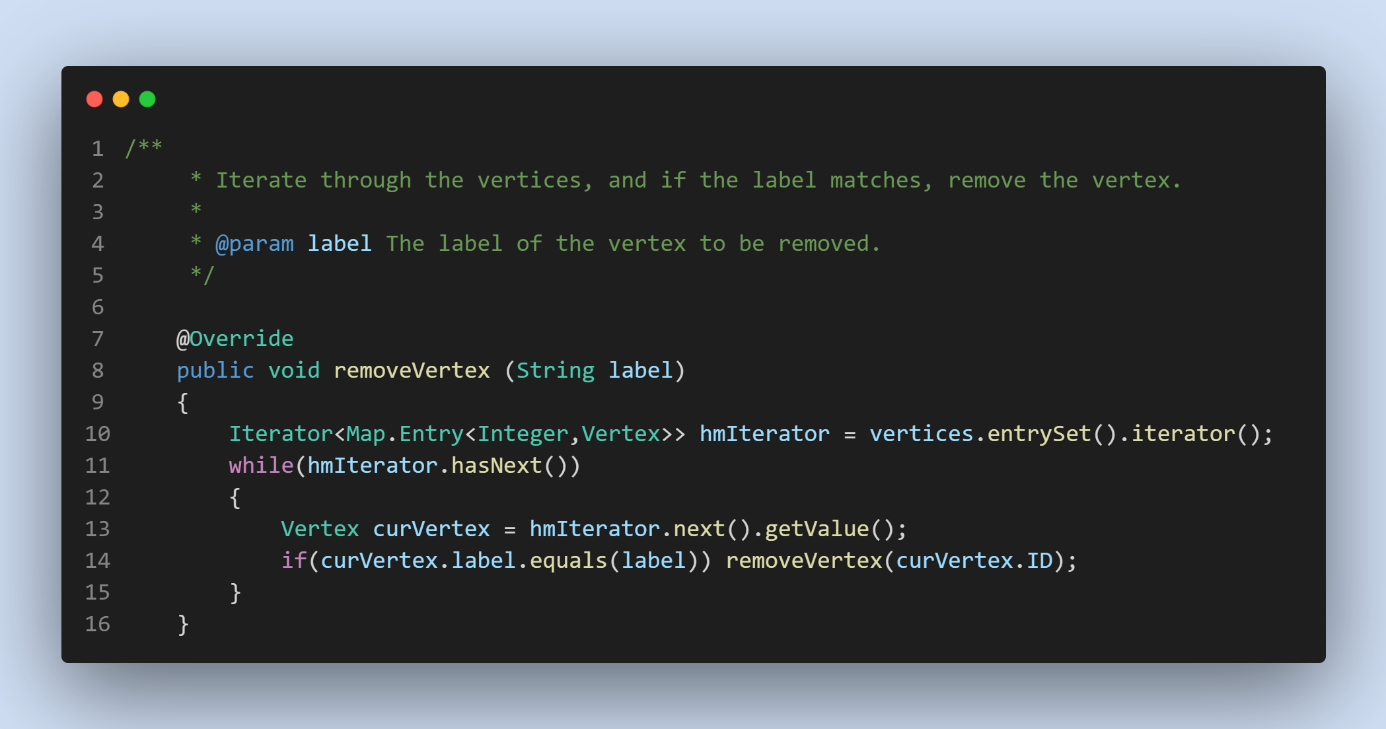
5-) Vertex removeVertex (int vertexID);

Complexity O(n\*m).



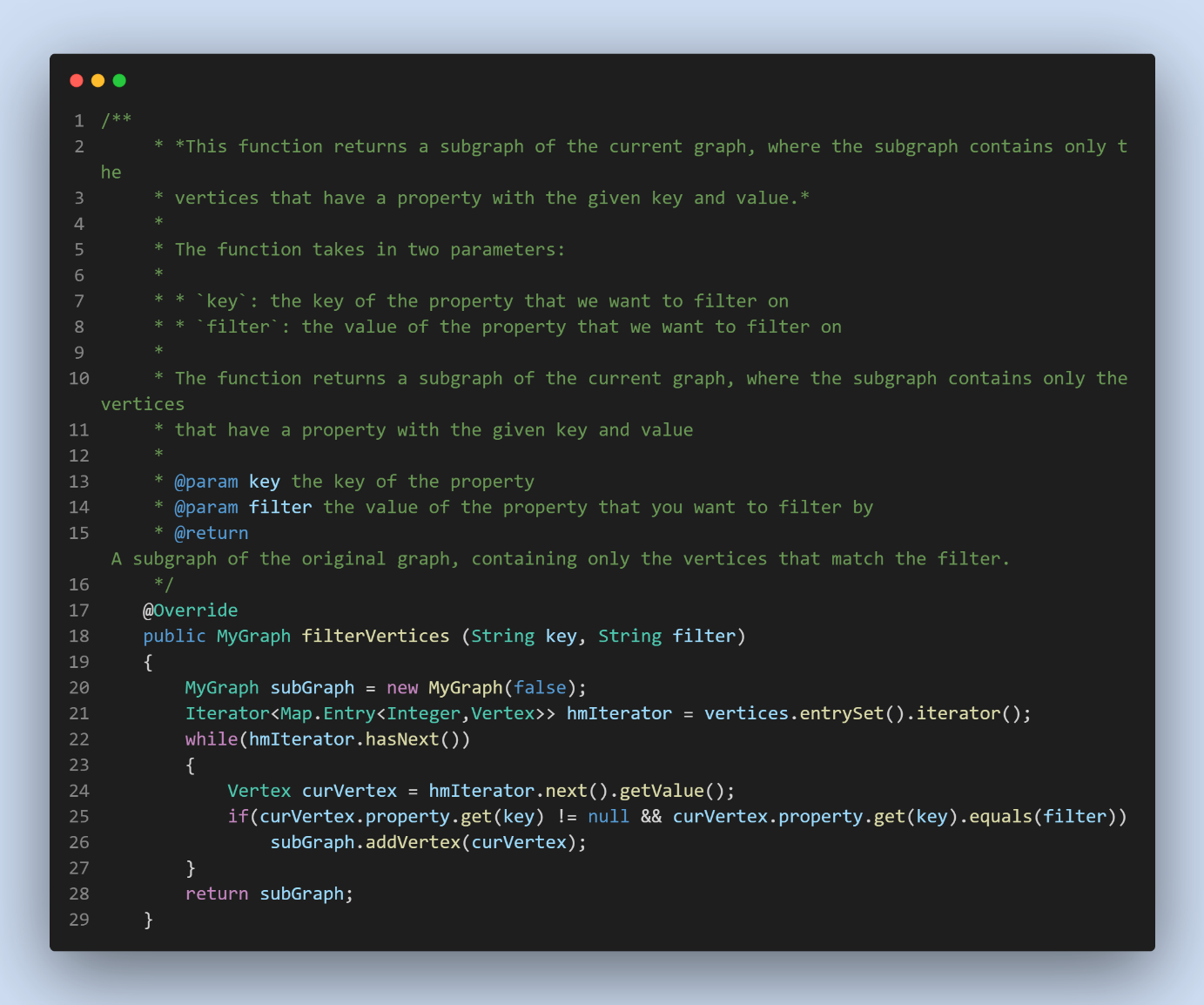
6-) void removeVertex (String label);

Complexity O(n\*m \* n ) = O(n^2\*m).



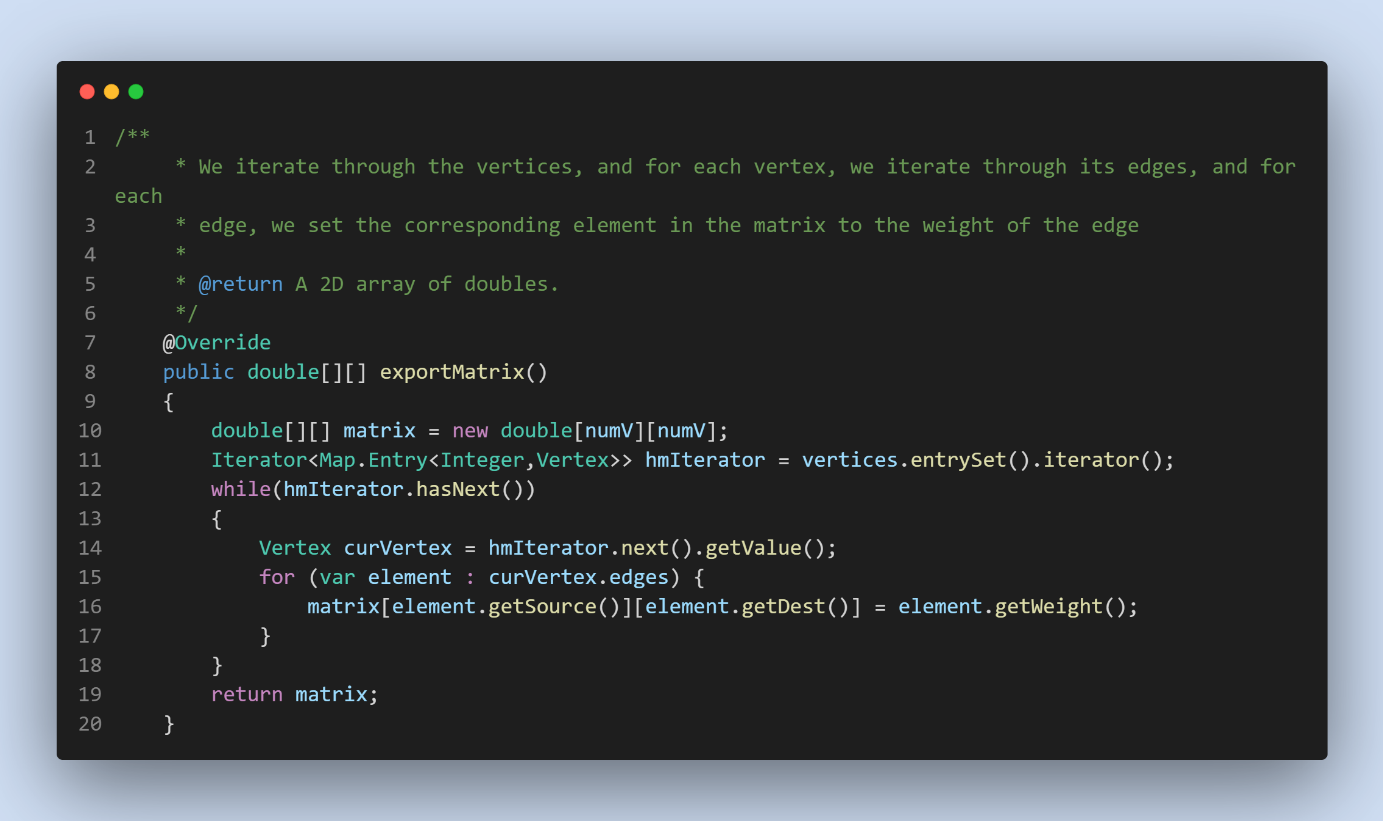
7 -) MyGraph filterVertices (String key, String filter);

Complexity O(n).



8-) double[][] exportMatrix();

Complexity O(n^2).



9-) void printGraph();

Complexity O(n^2).

