CS 231 A - Data Structures and Algorithms - Project 09 - Game: Hunt the Wumpus

Chandrachud Gowda
12/13/2021

Abstract

The core idea behind this project is to implement a graph and build a game i.e. Hunt the Wumpus that uses a graph data structure as a fundamental component. Here, the implemented graph will represent the layout of the game world. The additional object of this project is to implement Dijkstra's algorithm for computing the shortest path from a node to other nodes in a graph. The design approach I have used for representing the graph is using an adjacency list of each vertex, which holds all the connections leading that vertex.

The Graph class includes the addUniEdge( ) function which creates a uni-directional link between 2 vertices, the addBiEdge( ) function which creates a bi-directional link between 2 vertices, and the shortestPath( ) function which implements a single-source shortest-path algorithm i.e. Dijkstra's algorithm for the graph.

The HuntTheWumpus class is the main game controller class which contains the UI elements as well as fields for the Graph, Hunter and Wumpus. The class uncludes functions which listen for keystrokes using the *wsad* functionality for moving around the graph. The space bar arms the Hunter's arrow and the next keystroke (i.e. the wasd key) determines which direction to shoot the arrow in. Hitting the spacebar a second time without hitting one of the wasd keys rearms the arrow and enables the Hunter to move again.

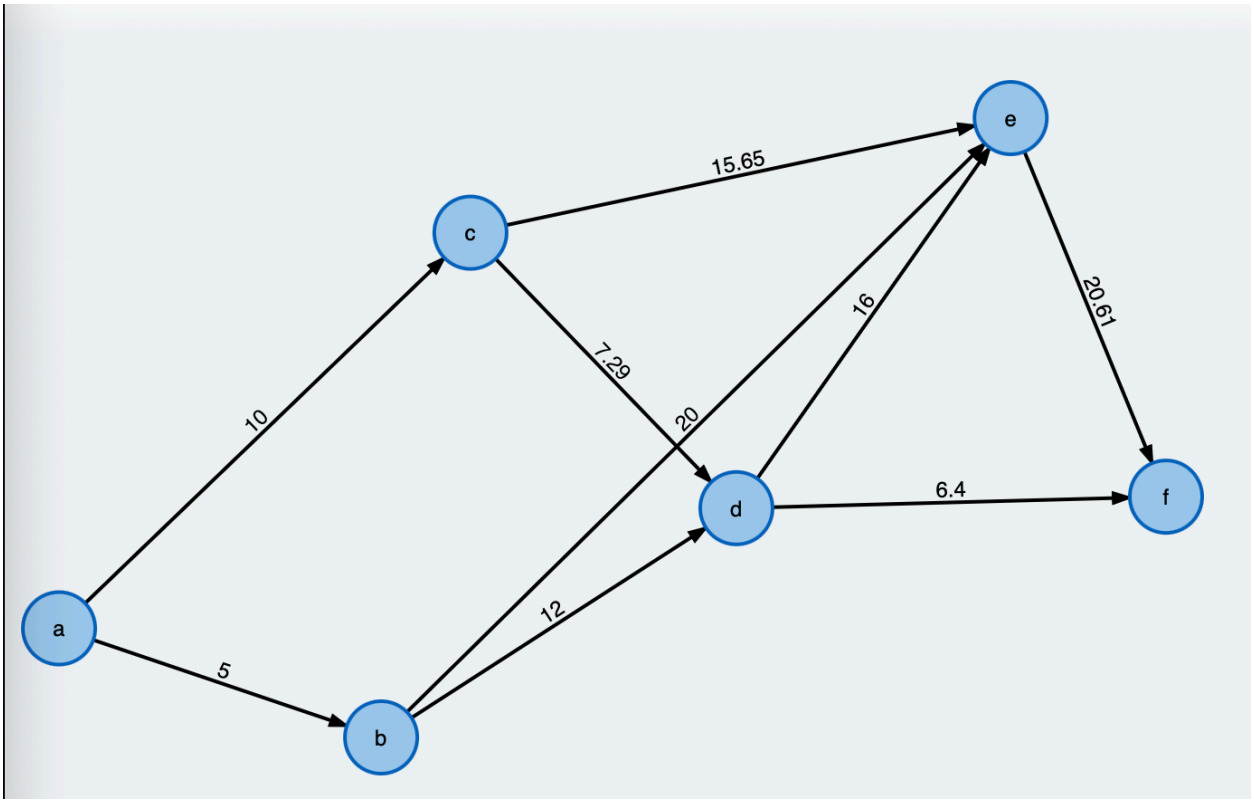
The interconnection of classes can be represented as follows: LinkedList.java, PQHeap.java, Vertex.java -> Graph.java, GraphTester.java, Hunter.java, Wumpus.java -> Landscape.java -> InteractiveLandscapeDisplay.java, HuntTheWumpus.java.

The key purpose of the project was to understand and become more familiar with the object-oriented approach of writing Java programs, the concept of classes and their interactions with each other, as well as understanding the core data structure behind this class - Graphs (here, using the adjacency list implementation).

Results

This section will highlight the results of my Graph.java and HuntTheWumpus.java classes.

I have chosen a sample graph to demonstrate that the Graph data structure and the shortest path algorithm work correctly. The graph can be represented as follows:

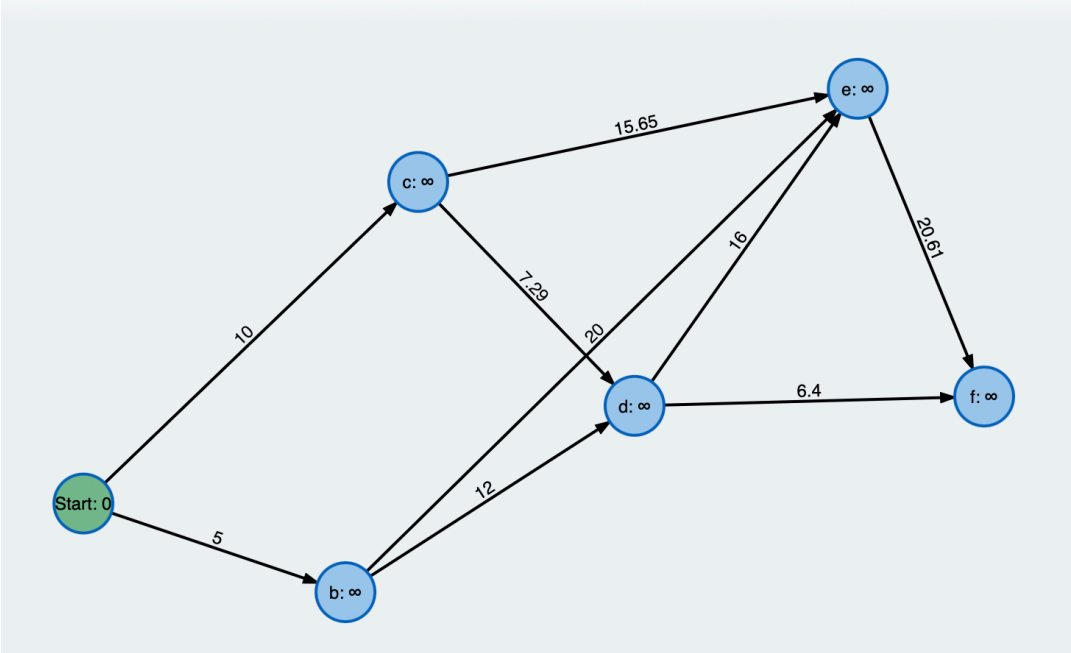


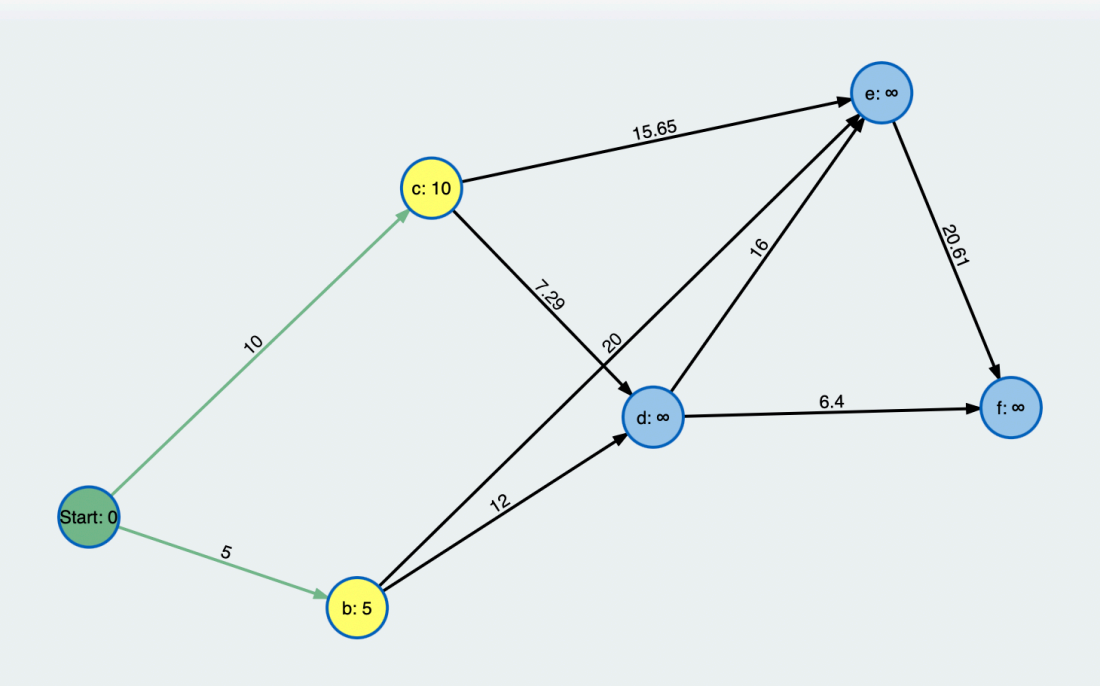
The adjacency list of the vertices can be represented as follows:

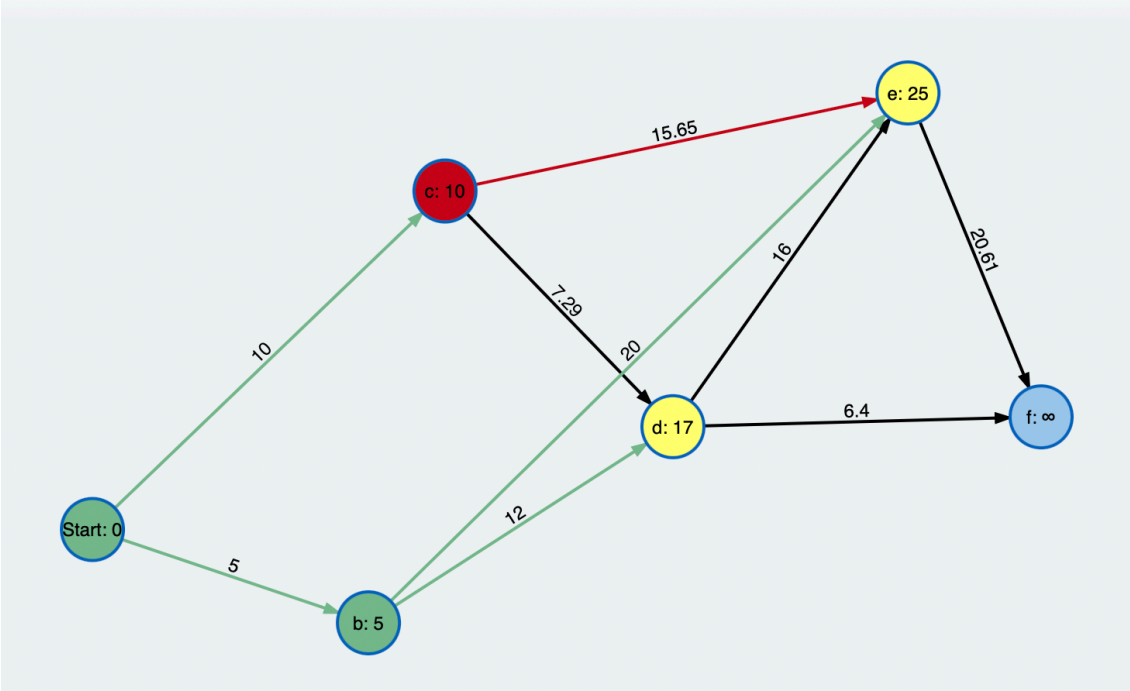
Vertex a: c (weight = 20), b (weight = 10)
Vertex b: e (weight = 10), d (weight = 50)
Vertex c: e (weight = 33), d (weight = 20)
Vertex d: f (weight = 2), e (weight = 20)
Vertex e: f (weight = 1)
Vertex f: -

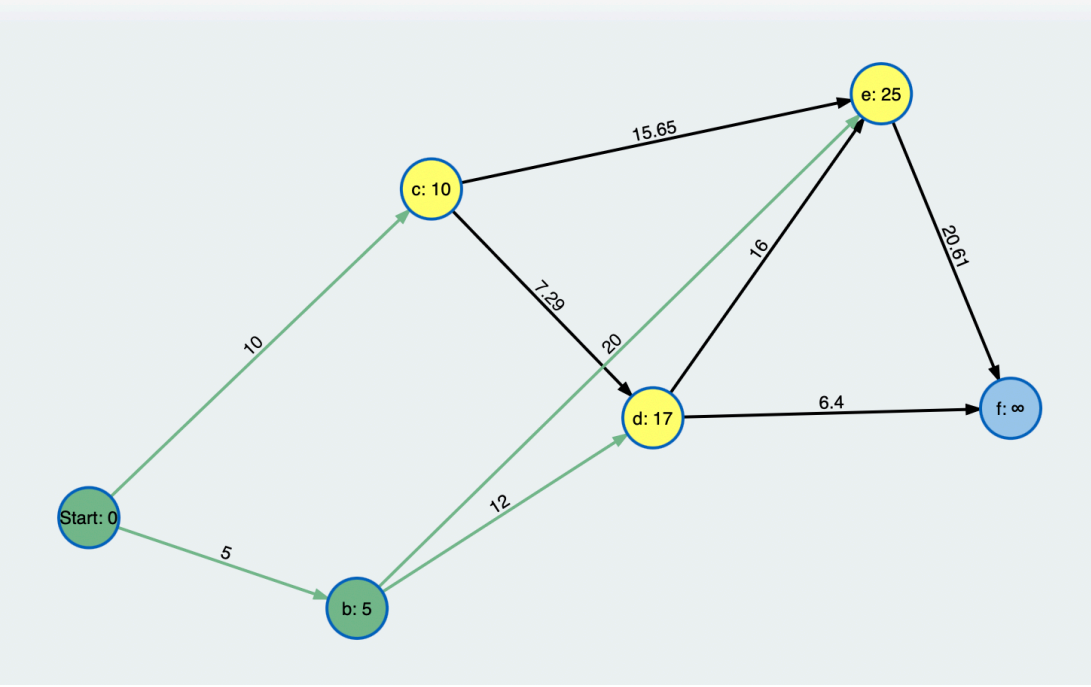
Starting vertex - Vertex a

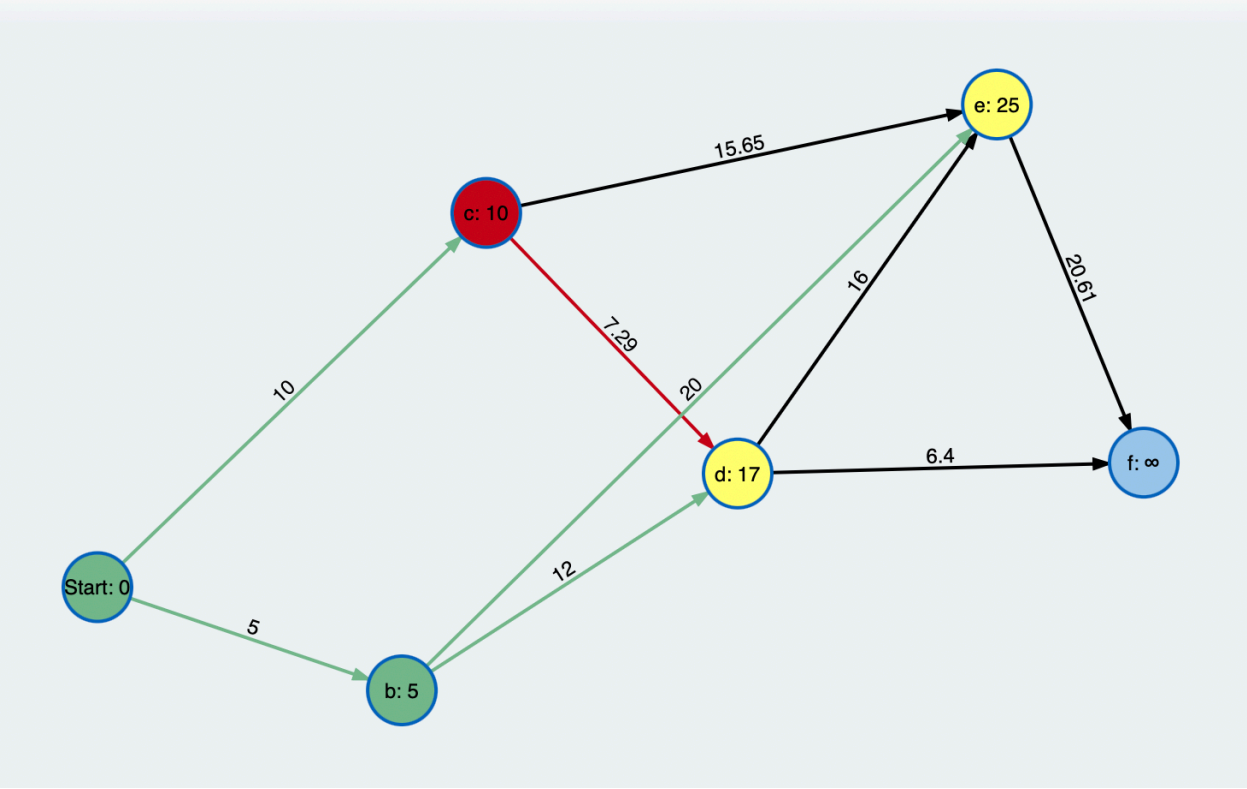
The execution of Djikstra's algorithm on the above graph can be represented as follows:

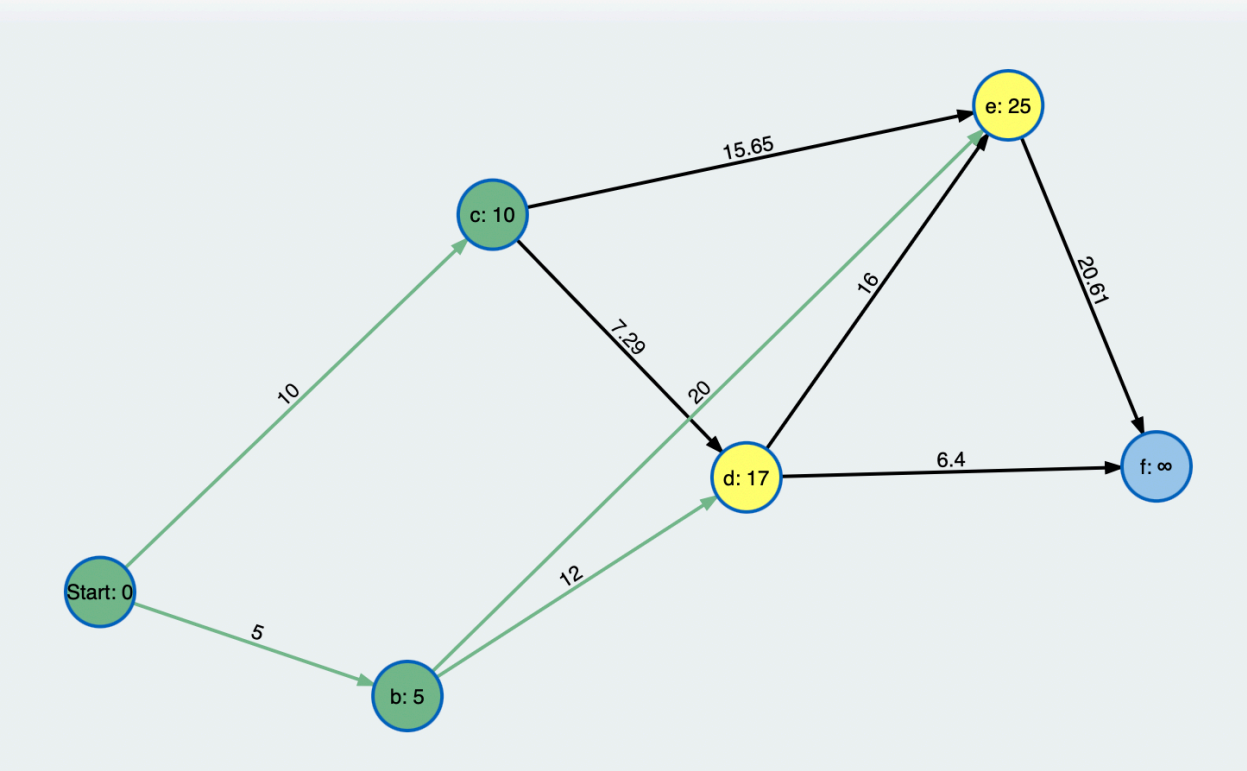


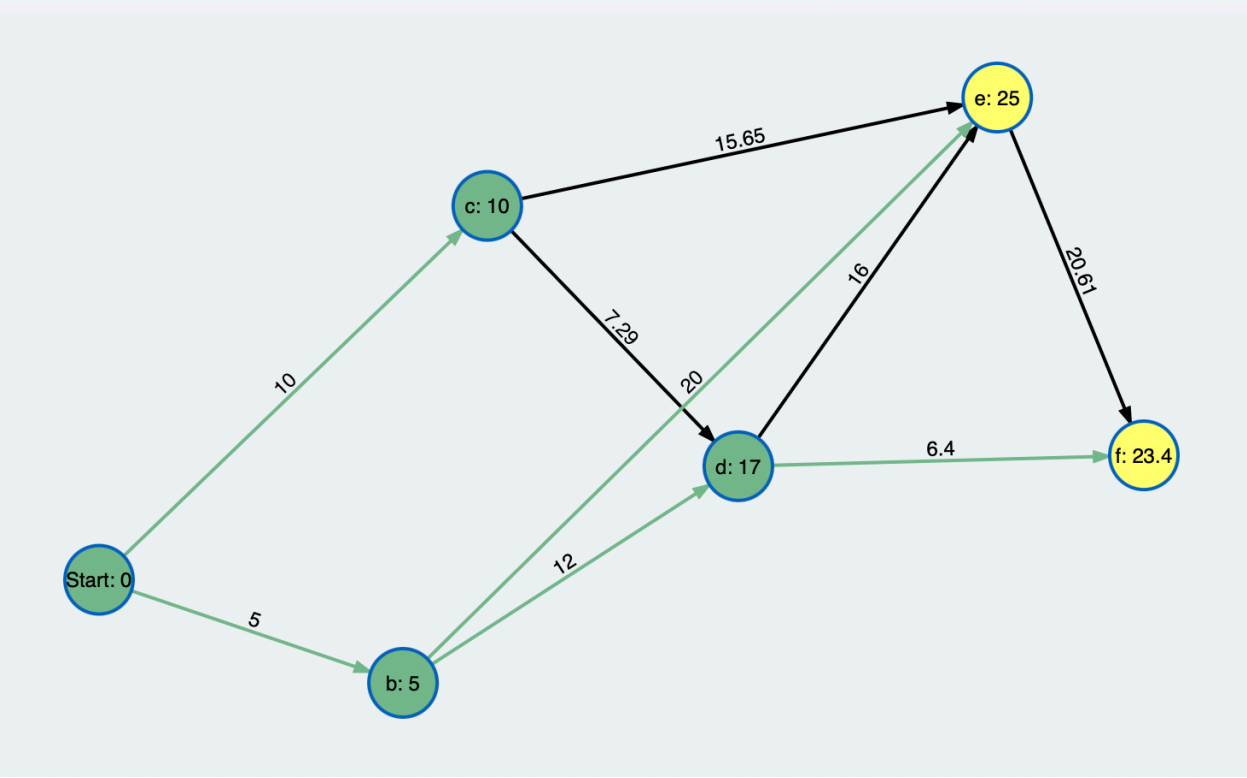


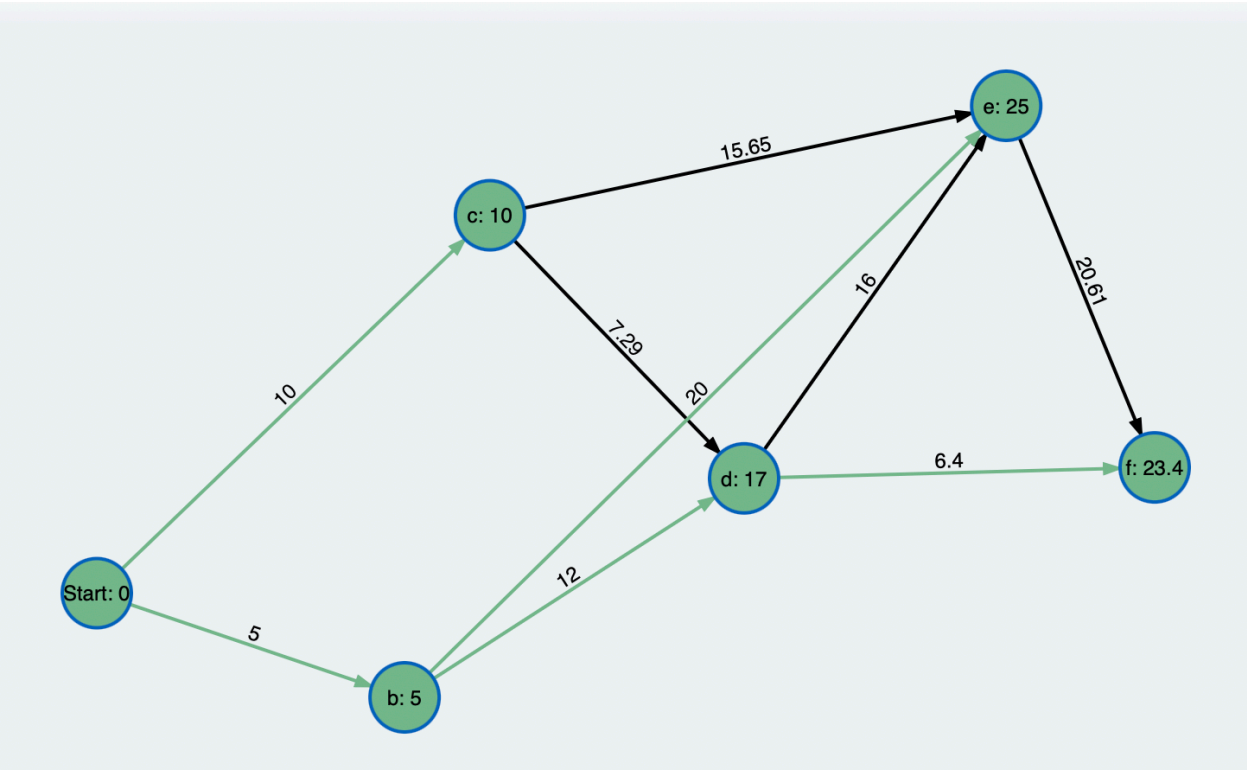

e: 25
15.65
c: 10
20.61
16
7.29
20
10
d: 17
6.4
f: ∞
12
Start: 0
5
b: 5


e: 25
15.65
c: 10
20.61
16
7.29
20
10
d: 17
6.4
f: ∞
12
Start: 0
5
b: 5


e: 25
15.65
c: 10
20.61
16
7.29
20
10
d: 17
6.4
f: ∞
12
Start: 0
5
b: 5


e: 25
15.65
c: 10
20.61
16
7.29
20
10
d: 17
6.4
f: ∞
12
Start: 0
5
b: 5


e: 25
15.65
c: 10
20.61
16
7.29
20
10
d: 17
6.4
f: 23.4
12
Start: 0
5
b: 5


e: 25
15.65
c: 10
20.61
16
7.29
20
10
d: 17
6.4
f: 23.4
12
Start: 0
5
b: 5

The result of the execution of the main method of the Graph.java function (with the above graph is as follows:

```
Before Shortest Path:
Number of neighbours: 2
Vertex cost: 0.0
Vertex coordinates: (0, 0)
Has the vertex been visited? false

Number of neighbours: 2
Vertex cost: 0.0
Vertex coordinates: (3, 4)
Has the vertex been visited? false

Number of neighbours: 2
Vertex cost: 0.0
Vertex coordinates: (8, 6)
Has the vertex been visited? false

Number of neighbours: 2
Vertex cost: 0.0
Vertex coordinates: (15, 4)
Has the vertex been visited? false

Number of neighbours: 1
Vertex cost: 0.0
Vertex coordinates: (15, 20)
Has the vertex been visited? false

Number of neighbours: 0
Vertex cost: 0.0
Vertex coordinates: (20, 0)
Has the vertex been visited? false

After shortest path:

Parent of Node A should be nothing : true
Cost of Node A should be 0 : 0.0
Parent of Node B should be Node A : true
Cost of Node B should be 5 : 5.0
Parent of Node C should be Node A : true
Cost of Node C should be 10 : 10.0
Parent of Node D should be Node B : true
Cost of Node D should be 17 : 17.0
Parent of Node E should be Node B : true
Cost of Node E should be 25 : 25.0
Parent of Node F should be Node D : true
Cost of Node F should be 23 : 23.40312423743285
```

I also implemented the HuntTheWumpus game. A sample demonstration of the game with a sample graph is included in the “Videos” section of the Google drive.

Extensions

Extension 1: Adding a Replay button and a Pause button

For this extension, I've implemented a Replay button and a Pause button for the HuntTheWumpus game. I added extra fields to the PlayState enum - PAUSE, FINISHED and REPLAY to indicate the extra states of the game to check for. Then I proceeded to add an if block around the control keys (including the firing keys) for when the pause button is pressed. Also, an if block was added inside the while block in the main function to check when the replay button is clicked, which further called the restart( ) function to restart the game by re-initializing all the parameters.

Acknowledgements

I haven't worked with any TAs or instructors for this project. I have not seen or referred to anyone else's code for this project. I used the following online resources:

https://docs.oracle.com/en/java/
https://en.wikipedia.org/wiki/Graph_(abstract_data_type)
https://docs.oracle.com/javase/7/docs/api/java/awt/Color.html
https://docs.oracle.com/javase/7/docs/api/javax/swing/JButton.html
https://docs.oracle.com/javase/7/docs/api/javax/swing/JPanel.html
https://docs.oracle.com/javase/7/docs/api/javax/swing/JLabel.html
https://en.wikipedia.org/wiki/Dijkstra's_algorithm