Okeoghene Excel Omorobe

CMSC 204

Assignment 6 Documentation

**Learning Experience**

I gained knowledge of how to represent real-world entities such as towns and roads using object-oriented programming within Java. Designing the Town class and the Road class solidified my concept of encapsulation as well as equality contracts (equals(), hashCode(), and compareTo()). I was also exposed to working with collections such as ArrayList and HashSet to handle dynamic data. Clean code with test cases ensured that I was able to test logic using JUnit, and I saw how clean, properly designed classes are the foundation of a bigger application.

Working with the Graph class using Dijkstra's algorithm was challenging but rewarding. It forced me to think about pathfinding, priority queues, as well as keeping records of distances and nodes. It strengthened my algorithmic mind, as it gave me a sense of confidence while performing graph theory. I learned to value efficiency as much as correctness, particularly when converting theoretical models into code.

The most important thing I learned from this assignment was the significance of testing and debugging. Writing both student composed and provided by the instructor JUnit tests helped me systematically test each element of the system. On failures of test cases, I learned to trace logic errors, review object comparisons, and ensure that data matched perfectly at various layers. These are essential skills to have when dealing with professional development environments where maintainability, reliability, and traceability of bugs are paramount.

**Pseudocode**

* **Class Town:**

Properties:

name : String

(Optional) adjacentTowns : Set<Town>

Constructor(name):

Set name

Method getName():

Return name

Method equals(Object other):

Return true if other is a Town and names match

Method hashCode():

Return hash of name

Method compareTo(other):

Compare by name

Method toString():

Return name

* **Class Road:**

Properties:

source : Town

destination : Town

name : String

weight : int

Constructor(source, destination, weight, name):

Set all properties

Constructor(source, destination, name):

Set weight = 1

Set other properties

Method getSource():

Return source

Method getDestination():

Return destination

Method getWeight():

Return weight

Method getName():

Return name

Method contains(town):

Return true if town is source or destination

Method equals(Object other):

Return true if both towns match, regardless of order

Method compareTo(other):

Return comparison of road names

Method toString():

Return formatted road string (optional)

* **Class Graph:**

Properties:

vertices : Set<Town>

edges : Set<Road>

adjacencyMap : Map<Town, Set<Road>>

Method addVertex(v):

If v not in vertices:

Add to vertices

Initialize adjacencyMap[v]

Return true

Else return false

Method addEdge(v1, v2, weight, description):

If both vertices exist:

Create road

Add to edges

Add road to both adjacencyMap[v1] and [v2]

Return road

Else throw IllegalArgumentException

Method removeEdge(v1, v2, weight, description):

Find road with given criteria

If found:

Remove from edges and adjacencyMap

Return road

Else return null

Method removeVertex(v):

Remove all edges from adjacencyMap[v]

Remove v from vertices

Method containsEdge(v1, v2):

Return true if such an edge exists

Method getEdge(v1, v2):

Return edge if it exists, else null

Method edgesOf(v):

Return all roads in adjacencyMap[v]

Method vertexSet():

Return vertices

Method edgeSet():

Return edges

Method shortestPath(source, destination):

Call dijkstraShortestPath(source)

Build path using previous node map

Format strings as required (Town\_A via Road\_X to Town\_B 4 mi)

Method dijkstraShortestPath(source):

Initialize:

distances = Map<Town, Integer> (infinity)

previous = Map<Town, Town>

visited = Set<Town>

priorityQueue ordered by shortest distance

distances[source] = 0

Add source to priorityQueue

While priorityQueue not empty:

current = poll from queue

For each neighbor via road:

If not visited:

Calculate new distance

If shorter:

Update distance and previous

Add neighbor to queue

Store previous map for path retrieval

* **Class TownGraphManager:**

Property:

graph : Graph

Method addTown(name):

Create new Town and add to graph

Method addRoad(t1, t2, weight, name):

Create edge in graph

Method getTown(name):

Search vertexSet for town with matching name

Method getRoad(t1, t2):

Return road name if road exists

Method allTowns():

Return sorted list of town names

Method allRoads():

Return sorted list of road names

Method containsTown(name):

Return true if town exists

Method containsRoadConnection(t1, t2):

Return true if road exists

Method deleteRoadConnection(t1, t2, roadName):

Find and remove the road

Method deleteTown(name):

Remove town and all connected roads

Method getPath(t1, t2):

Get path from Graph.shortestPath()