ECE 250 - Project 5
Shortest Path using Dijkstra's Algorithm
Design Document

Harshit Manchanda, UW UserID: h2mancha

Apr 24th, 2020

1. Overview

Class 1: Node

Description:
It is used to create a linked list in the undirectedGraph class which helps in storing the adjacent vertices of a vertex.

Member Variables:
1. Integer position
2. Node type pointer next (points to the next node)

Class 2: PriorityQueue

Description:
It is the Priority Queue class which is created to help find the shortest distance between two cities of the graph. It has functions which are used in Dijkstra's algorithm to find shortest distance like extract minimum to get the minimum valued vertex from the queue and find the position of a vertex in the queue.

Member Variables:
1. A vector queue of type vertexValues (a struct)
2. Integer size

This vector represents the priority queue and stores the vertices with their distances from source.

Member Functions (operations):
1. extractMin: it returns the minimum valued node in the priority queue.
2. searchPosition: it returns the index of the node sent as parameter in the queue.

Class 3: undirectedGraph

Description:
It is a weighted undirected graph represented by an adjacency matrix which stores information about the edges between different vertices and their weights. It has functions like insert edge, set distance, finding the degree of a particular vertex, finding the shortest distance between two vertices which is achieved using the Priority Queue class, list vector and Set S etc.

Member Variables:
1. Integer nodes which holds the number of nodes in the graph
2. Integer edges which holds the number of edges in the graph
3. An adjacency matrix to keep record of the edges and their weights
4. A string vector to store the name of the cities
5. A vector of type List(struct) to store the adjacent vertices
6. A vector of type vertexValues to store the shortest distances from a source in the graph.

Member Functions (operations):
1. insertNode: insert a node everytime the function is called in the matrix and also adds a node in the list vector.
2. setDistance: checks if the vertices passed as parameters are in the valid range, if so, then adds that edge between them in the matrix with its weight. It also adds that vertices in the list vector to keep a record of respective adjacent vertices.
3. searchCity: returns the index of the city in the matrix if it exists.
4. degreeOfCity: returns the number of number of vertices a vertex is connected to by checking in the adjacency matrix.
5. numberOfNodes: returns the value of the member variable nodes which holds the number of nodes
6. numberOfEdges: returns the value of the member variable edges which holds the number of edges
7. printDistance: returns the edge weight between two vertices from the adjacency matrix.
8. shortestDistance: it calculates the Shortest distance by applying Dijkstra' Algorithm where the Priority Queue holds the distance from the source vertex and the Set S keeps storing the added minimum values from the queue.

9. printPath: it prints the Shortest distance by applying Dijkstra' Algorithm where the Priority Queue holds the distance from the source vertex and the Set S keeps storing the added minimum values from the queue.
10. clearGraph: it deletes the whole graph by emptying all the vectors and the adjacency matrix

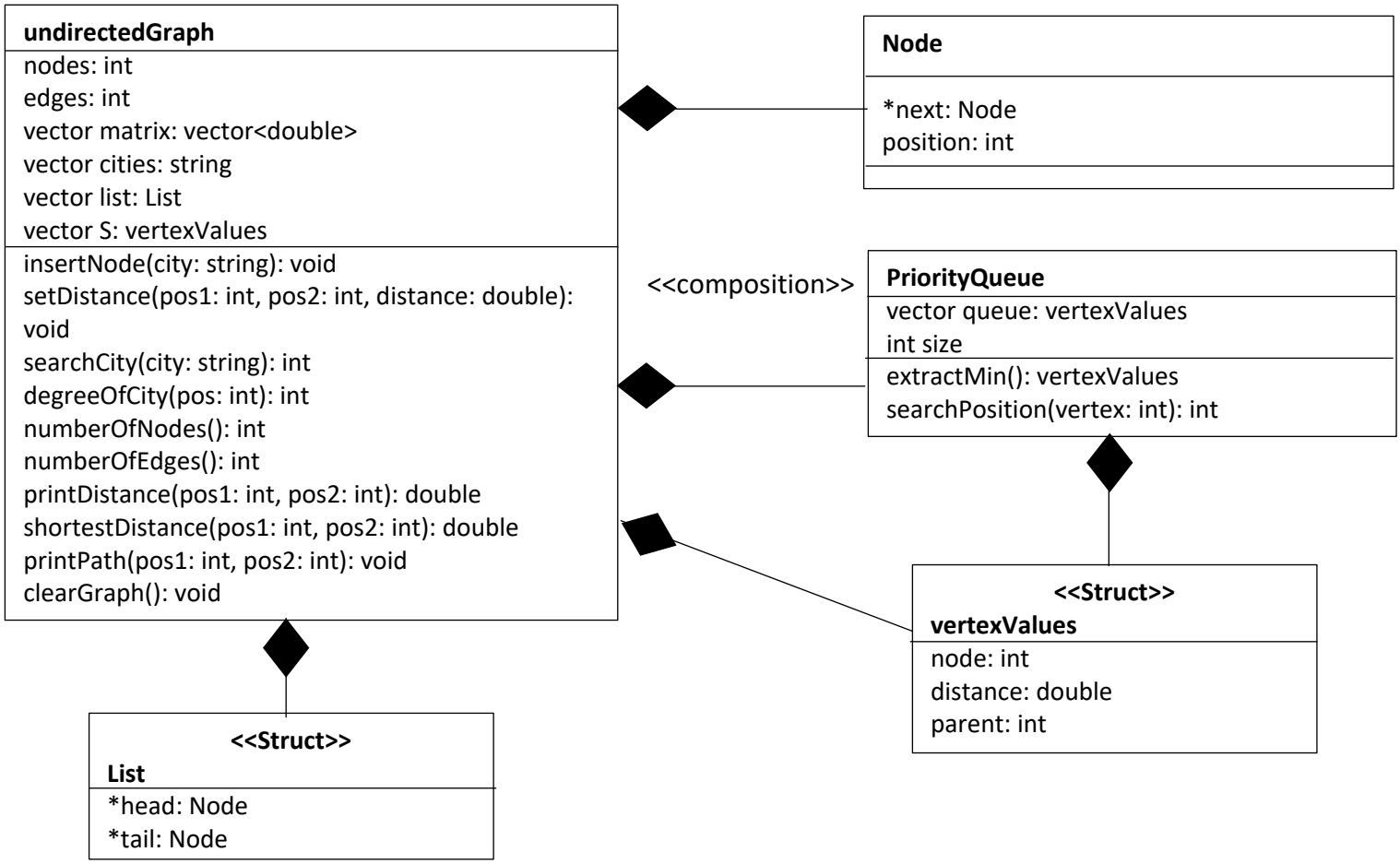
Struct 1: vertexValues
It helps create a vector(priority queue) which will store the distance, vertex and parent to help implement Dijkstra's algorithm.

Member Variables:
1. A double distance which stores the distance from the source
2. Integer node storing the vertex
3. Integer parent which stores the parent of that vertex

Struct 2: List
This struct also helps create a vector in the undirectedGraph class which stores adjacent vertices of a vertex.

Member Variables:
1. A node type pointer head
2. A node type pointer tail

UML Class Diagram -



2. Constructors/Destructors

Class Node (Constructor and Destructor):
This class's parametrised constructor takes in the vertex and address of the next pointer and assigns them to the member variables of that node object. For releasing the memory, we have a destructor in the undirectedGraph class which helps in freeing the memory.

Class PriorityQueue (Constructor and Destructor):
This class has no member which consist of dynamically allocated memory, there is no need for a constructor or destructor.

Class undirectedGraph (Constructor and Destructor):

The matrix is already a member variable so there is no need to initialise it and in the case of the destructor, since we had allocated memory manually for the list vector, we need to have a destructor which releases and frees that memory.

3. Test Cases

Test 1: Create a new graph, insert nodes, set the distance between some vertices and check the number of edges. Also check if the degree of a vertex is correct. Now clear the graph and again check for the number of edges.

Test 2: Create another graph, insert nodes and calculate the shortest distance between some nodes. Check if the calculated distances are correct or not. Now, clear the graph and check if the distance is calculated or not

Test 3: Now, create another graph with all valid edges such that it is connected. Calculate the distance between some cities and also print the path between them to check if it correct.

Test File Example 1:

```
i city0
i city1
i city2
setd city0;city1;9
setd city2;city1;8
graph_edges
degree city1
clear
graph_edges
```

Test File Example 2:

```
i city0
i city1
i city2
i city3
setd city0;city1;9
setd city2;city1;8
setd city3;city0;5
setd city2;city3;3.2
shortest_d city0;city3
clear
shortest_d city0;city3
```

Test File Example 3:

```
i city0
i city1
i city2
i city3
setd city0;city1;5.6
setd city2;city1;4.3
setd city3;city0;9.1
setd city2;city3;3.2
shortest_d city1;city3
print_path city1;city3
```

4. Performance

For class undirectedGraph:
The insertNode function resizes the matrix, so its time complexity is O(V^2) where V is the number of vertices because it's a VxV matrix.
The setDistance function runs in constant time O(1) cause it only consists of conditional statements and sets value in matrix.
The searchCity function also runs in linear time O(V) because it finds the index of the node from the cities vector by looping.
The degreeOfCity function consists of a for loop which runs through the row of that vertex in the matrix to count the edges, so it has a time complexity of O(V).
The numberOfNodes function simply returns the value of the member variable so is implemented in constant time O(1).
The numberOfEdges function simply returns the value of the member variable so is implemented in constant time O(1).
The printDistance function just returns the value of that edge from the matrix so it is also implemented in constant time O(1).
The clear edges function calls the clear function of the vector and thus we get a time complexity of O(V^2).

Now for the function shortestDistance, we have implemented Dijkstra's algorithm using Priority Queue. First we create one by using a for loop and add all the nodes with the value infinity except the source which has the value 0. Then, we have nested loops in which the first loop runs V times and calls the function extractMin from the priority queue which runs in O(lgV) and the second loop iterates through the adjacent edges and enters the value if relaxation is possible. That also takes time O(logV). So finally, we get the time to be O(ElgV) which is the desired time.
Similar is the case for print path which also uses Dijkstra's algorithm to get the distances and then we use the parents of the city to which we need to find the path to in order to print it. It also takes a time of O(ElgV).