**CMPE 343 HW 4 Report  
4865115490,  
Fatih Mehmet Yiğitel,  
Section 02,  
HW4**

Problem Statement and Code Design:

1st Task:

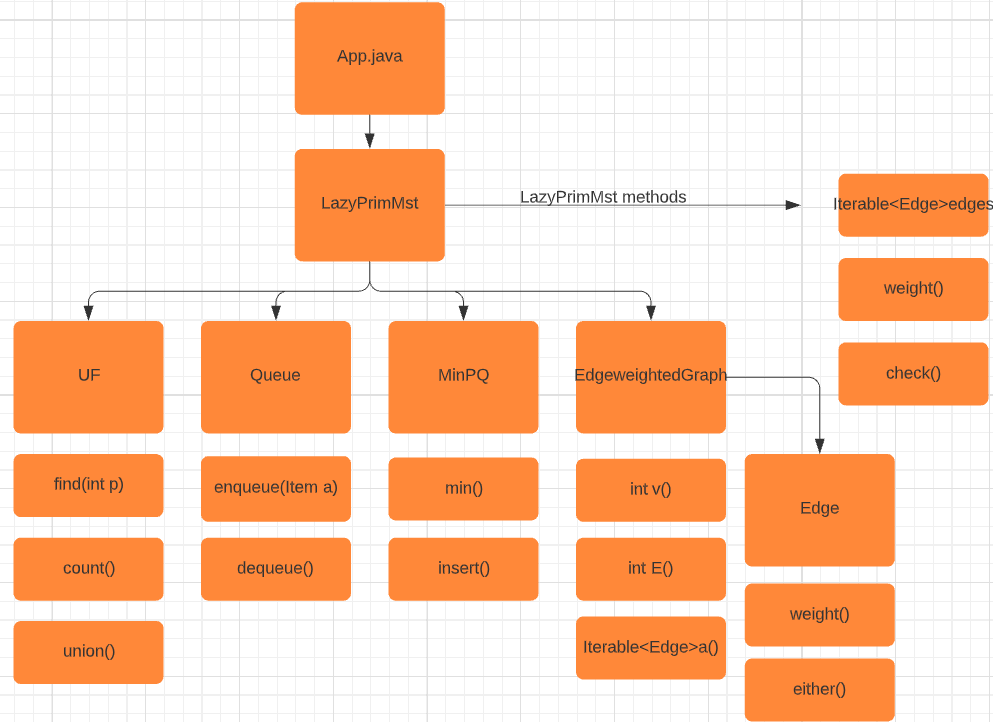
In this part we must find all of the short ways to that have access every city. I built highspeed railway in the USA. To do this we will implement MST. To implement an MST I use EdgeWeightedGraph. We have given cities and distances between cities in the txt file. Using these inputs we have to represent them as a graph at first. After that we use some of the mst algorithms and finding the shortest ways for each city. I use Lazyprim MST algorithm because it was the closest one for the output. At the and in the main class I implement some codes for the needed output that have lexographical order and level order traversal. In addition I calculate the total distance.

2nd Task:

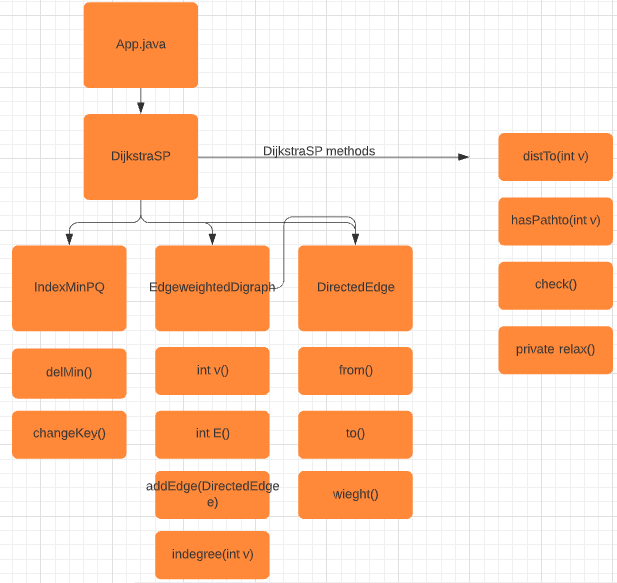
In the second task we have given cities, vertex number for cities, city coordinates and after that the city vertex that will be bonded with directed edges. In the task we are asking for find the smallest path between to city. To do that I use directedgraph class to create it and DijskstreSP class to find the closest distance. So, I take input them into arrays and create a edge weighted digraph. For the weight of edges, I have to use a distance formula (Euclidean distances). I with calculated it I represent the cities in a diagraph. Secondly with using DijsktraSP I find the closest path and I print how many cities we have visited, visited cities names, the total distance.

After that I use different inputs and understand how DijkstraSP works (time complexity) and report it. We use MinPQ to increase efficiency. I understand and prove that the complexity of Dijkstra O ( V + E l o g V ) .

Task 1 Diagram:



Task 2 Diagram:



Implementation and Functionality:

1st Task:

Main class: In the App.java I implement the I use scanners to take the cities as input. After that I hold the edge inputs in separate arrays. I take them line by line. For every line I create edges using the every input in my arrays. After creating the edges I hold them in “arrayedge” array. After that add them into my EdgeWeightedGraph using for loop.

for(int i = 0; i<arrayedge.length;i++){

    G.addEdge(arrayedge[i]);

}

Secondly I use LazyPrimMst algorithm. I create it with my EdgeWeightedGraph G

LazyPrimMST LP =new  LazyPrimMST(G);

And I hold the edges in the MST to arrange them later.

 Iterator<Edge> iterator =LP.edges().iterator();

        List<Edge> Listed = new ArrayList<>();

        iterator.forEachRemaining(Listed::add);

Using the Listed array I calculate the total weights of the edges in the mst

 double totalWeight = 0;

    for(int i = 0; i<Listed.size(); i++){

        totalWeight += Listed.get(i).weight();

    }

        System.out.println((int)totalWeight);

Thirdly I arrange the edges as lexographical order and level order traversal using some for and if loops

I use one for loop

 for(int i = 1; i<Listed.size();i++){

and additionally

if(arr[Listed.get(i-1).other(Listed.get(i-1).either())] == arr[Listed.get(i).other(Listed.get(i).either())]){

and some if statements with using other and either methods.

Eventually I get correct result.

LazyPrimMST class:

Constructor:

It has Queeue and Mininum Priority queue data structure for holding edges. Min priority queue increase efficient in terms oof time complexity.

We have marked array for holding visited cities. We visit every city to compute mst. I n the for loop we call prim algorithm if there is marked vertex

 for (int v = 0; v < G.V(); v++)

 if (!marked[v]) prim(G, v);

It computes a minimum spanning tree of an edge-weighted graph.

public Iterable<Edge> edges()

it returns the edges in a minimum spanning tree as iterable.

public int weight()

it returns the sum of the edge weights in a minimum spanning tree.

Edge class:

Constructor():

public Edge(int v, int w, double weight) {

In the constructor we have two vertex number as parameters and a double value for edges weights. We first check vertex numbers are valid.

public int either()

It returns either endpoint of this edge.

public int other(int vertex)

It returns the endpoint of this edge that is different from the given vertex.

2nd Task:

Main class:

Firstly, I use scanner for the take input as file name, source city name, destination city name.

I use “cities” array for the all the lines with an order, “edges” array for the holding the bounded two vertexes numbers with an order.

String cities [] = **new** String[numvertices\*4];

I multiply numvertices with four because every line has 4 element.

String edges [] = **new** String[numedges\*2];

I multiply numedges with 2 because every edge consist of 2 vertex.

Finally I hold the all the values in the arrays.

I create two array for the X coordinates and Y coordinates and City array for keeping city’s names.

Using cities array I separated required coordinates these three array.

**for**(**int** i = 1; i<cities.length; i=i+4){

Xcordinates[c] = Integer.*parseInt*(cities[i]);

c++;

}

I create source and destination arrays to keeping the source and destination vertexes and after using creating edges for the graphs.

**int** [] source = **new** **int**[numedges];

**int** [] destination = **new** **int**[numedges];

**for**(**int** i = 0; i<numedges\*2; i=i+2){

source[b] = Integer.*parseInt*(edges[i]);

destination [b]= Integer.*parseInt*(edges[i+1]);

b++;

}

Using edges array I separate the vertex according to their situation that source or destination.

After that I use these arrays to find the distance between two vertex.

dist = ((Xcordinates[source[i]]Xcordinates[destination[i]])\*(Xcordinates[source[i]]-Xcordinates[destination[i]])+(Ycordinates[source[i]]Ycordinates[destination[i]])\*(Ycordinates[source[i]]-Ycordinates[destination[i]]));

**\*Euclidean distances formula**

After every elements rearranged for the EdgeweightedDigraph we add our created edges Edges arraylist. I use arraylist for edges because it is easy to add something in arraylist.

Edges.add(i ,**new** DirectedEdge(source[i],destination[i],last));

After that we add it to our EdgeWeightedDigraph G.addEdge(Edges.get(i));

I convert City array to cityArrayList to use them easily.

After EdgeWeightedDigraph created completely I create MST object

DijkstraSP DJ = **new** DijkstraSP(G,a1);

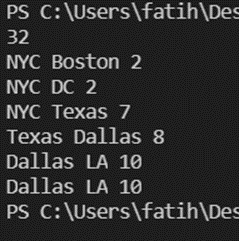
a1 is the source vertex index number. we set parameters. Using path to method I find the how to reach destination vertex. It is iterator method so I convert it list. Using the created list named “needed” I find them the cities between source and destination. Length of this list gives us how many city that we visited. And I print it correctly. I coludnt get it first so I reverse to reach correct output. At the end I find the total distances with using DistTo() method and I print them also.

Testing:

1st Task:



I test my main class for exceptions If I use different inputs. When I changed only weights it gives correct total weights.



But when I change some of the city names my last line is not give correct result.

NYC Texas LA Boston DC SF Dallas

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu



Also the total weights calculation works for the negative weights

2nd Task

I give it an additional file.

Final Assessments:

1st Task:

Making lexographical order and level order traversal was hardest part for me. It is hard to get correct output for them.

Using either and other methods for getting edges vertex is the most challenging part for me . I tried to all the MST algorithms in the book.

Observing how the mst algorithm, basic maps applications works was good. I learned how to find the appropriate way to go somewhere.

2nd Task

I think the hardest part is how to place the different type of inputs in arrays

In this task the challenging part was making a tester class and designing output for testing phase and observing the how algorithm works.

Testing and learning the complexity and behavior of the Dijsktra algorithm is also the interesting part for me.