Coordinate Class:

I wrote this class to make the working with the points in the map easier. The class has X and Y variables that represents the coordinates of the point and simple methods like getters. Class also has functions like one that returns a boolean value represents whether a given point is the neighbour of ‘this’ point and one that returns the Array List of the all the neighbour coordinates. And the class also have some basic overridden functions like .equals, .hashCode, .toString.

MapToPng Class:

I wrote this class to make a jpeg from a 2D int array in which the elements in it: 0 represents empty, 1 represents non-empty, 2 represents path. The static convert function of this class paints the jpeg file accordingly by iterating the 2D array.

CSE222Map Class:

This class works as specified in the pdf, the constructor takes the file argument and separates each line (except the first one and second one that represents the start and the end point) in a loop until the file end, and separates the line by ‘,’ using split method in java, and enters a new loop for each element of the separated line. Then the function uses the iterator variables as the coordinate value and the separated line’s iterated value as whether that location is occupied or not to construct the <coordinate, boolean> hashmap. This constructor also generates a map array that can be used as input to the algorithms that constructs the roads to prevent multiple construction of the map array.

CSE222Graph Class:

This functions has a constructor that receives a map as input, from this map it constructs the nodes. Each node is a defaultmutabletreenode that contains a Coordinate and children. The class has a coordinate hashset to keep track of the already added coordinates to the tree, and also has a hashset that contains all the nodes. Fillgraph function at the end of the constructor takes already created root node(from the start point) as argument and constructs the remaining nodes as neighbour coordinates from the root node(by checking whether they are in the map and checking whether they are empty) and from the neighbour nodes from the root node... until the end Coordinate by using a basic stack algorithm. The class has basic getters and a getMapAsArray function that returns the map array from map class.

DijkstraAlgorithm Class:

This class takes a graph as constructor input and uses the graph to return the solution of the shortest road from start to end as a 2D int array. It uses the algorithm specified in the pdf, but since the weight of our graph is predetermined (all neighbour coordinates that are not-cross has sqrt(2) distance between them and all of the other neighbour coordinates has 1 distance between them.) I wrote a simple getDistance function that checks whether the given coordinate is cross or not and returns the distance accordingly to implement the algorithm. It is the same as the algorithm in the pdf (additional reconstruction of the path part to construct the path by traversing from end to start by determining the shortest path from the distance hashmap that is constructed in the previous part.), I changed the equality operator parts to “difference is less than 0.0001” in order to prevent double number equality problems. Getsolution method uses the preconstructed 2D int array by map class and assigns the points that are in the path to 2 inside of the array.

BFS Algorithm Class:

This class implements a bfs algorithm that is in the pdf. General structure of the class is almost same as the DijkstraAlgorithm Class. Only issue is this class implements less accurate shortest paths because it won’t use the distance information between the coordinates like the dijkstra algorithm. Because all the children nodes from a coordinate node are the neighbour coordinates and they won’t know whether they are cross or not.

WritePath Class:

This class has 1 simple function that iterates HashSet of path coordinates and writes them to a mapName\_solution.txt.

I implemented the algorithms without using the nodes that are constructed in the graph class to prevent the performance issues. The algorithms simply use Coordinate.neighbours function acting like children node.

I slightly changed the classes provided: Changed the “Map” to “map” in Main method and I added a string folderName that is “TextFiles“ into the TestClass to import txt files from folder named TestClass.