Assignment 4 Report

P1 Divine Ingenuity

The code defines a nested class named `coordinate` to represent coordinates in the domain. Each coordinate has attributes for x and y positions, along with a distance value (which indicates the distance between `i` and the current node). The `b` (`i`)and `f` (`j`) objects of the `coordinate` class represent the starting and destination points, respectively. The program uses a priority queue to explore coordinates in ascending order of distance. ``

The main logic of the program lies in the `findMinimal` function, which implements a modified Dijkstra algorithm to find the shortest path from the starting point 'i' to the destination point 'j'.

```
private static int findMinimal(char[][] domain, int m, int n) {
  boolean [][] visited = new boolean[m][n]; // to check if the node has
been settled
  PriorityQueue<coordinate> heap = new PriorityQueue<>
(Comparator.comparingInt(coordinate -> coordinate.distance));
  b.distance = 0;//initialize the begin position
  heap.add(b);
  while (!heap.isEmpty()) {
    coordinate t = heap.poll();
    if(visited[t.x][t.y]) continue; // check if t has been settled
    visited[t.x][t.y] = true;
    if(t.x == f.x && t.y == f.y){ // reach j (final position)
      return t.distance;
    }
    List<coordinate> neighbours = getNeighbours(domain,t); // get
neighbours of t (along with the assignment of their distance)
    for(coordinate neighbour : neighbours){
      if(!visited[neighbour.x][neighbour.y]){
        heap.add(neighbour);
      }
    }
  }
  return -1;
}
```

The algorithm uses a priority queue to manage the order of exploration, and it considers neighboring coordinates to determine the minimum distance. (If the moving direction is consistent with the wind direction, then the distance of this t's neighbor is t.distance. Otherwise, the distance is t.distance + 1). For neighbors of four directions (in the domain), update each distance using t.distance and add it to the neighbors' lists. Then add to the priority queue if it hasn't been settled.

P2 Edge Changing

For this question, it requires two operations. Add the edges according to the conditions and print by breath first search. First, I design a Node class that only has an ArrayList neighbor.

When receiving inputs, I noticed that there might be multiple edges with different directions, however, the question doesn't need to consider direction. Therefore, I do the following operations.

```
  for (int i = 0; i < m; i++) {
    a = sc.nextInt();
    b = sc.nextInt();
    if (graph[a].neighbours.contains(b)) {
      continue;
    } else {
      graph[a].neighbours.add(b); // add the edge to each adjacent list
      graph[b].neighbours.add(a);
    }
  }
```

Then, for adding edges, traversal all the nodes. For each node,

```
//traversal all the neighbours
int neighbourLength = graph[i].neighbours.size();
for (int j = 0; j < neighbourLength; j++) {// compare each two neibours
  for (int f = j + 1; f < neighbourLength; f++) {
    if (graph[i].neighbours.get(j) == k * graph[i].neighbours.get(f)
//satisfy the condition
        || graph[i].neighbours.get(f) == k *
graph[i].neighbours.get(j)) {
      if
(graph[graph[i].neighbours.get(j)].neighbours.contains(graph[i].neighbo
urs.get(f))) {    // if the edge has existed, do not add it again
        continue;
      } else {
        //add the edge to each vertix adjacent list

 graph[graph[i].neighbours.get(j)].neighbours.add(graph[i].neighbours.g
et(f));

 graph[graph[i].neighbours.get(f)].neighbours.add(graph[i].neighbours.g
et(j));
      }
    }
  }
}
```

Finally, output.

```
private static void bfs(Node[] graph, int s, boolean[] visited) {
  LinkedList<Integer> queue = new LinkedList<>();
  queue.add(s);
  visited[s] = true;
  while (!queue.isEmpty()) {
    int v = queue.pollFirst();
    graph[v].neighbours.sort(Integer::compareTo);//output the neibours
in an acsending order
```

```java
        for (int neighbour : graph[v].neighbours) {
          if (!visited[neighbour]) {
            queue.add(neighbour);
            visited[neighbour] = true;
          }
        }
        if (v == s) System.out.print(s);
        else System.out.print(" " + v);
      }
    }
```