CSS533: Program 2

Shreevatsa Ganapathy Hegde

University of Washington, Bothell

sghegde@uw.edu

Prof. Munehiro Fukuda

1st May, 2025

Table of Contents

Documentation:

The Documentation section involves the implementation details of the Program 2: Flight Data Analysis with Apache Storm that I have built. This section contains the implementation of the requirements that were given in the program 2 document. The additional features will be discussed in the discussion section.

TopologyMain Class: This class sets up and executes a Storm topology locally for processing flight data. It uses a spout (FlightsDataReader) and connects it to two bolts (HubIdentifier and AirlineSorter) to group and analyze flights based on proximity to airports and airline information. The main() method accepts two arguments: the first is the path to flights.txt with flight data, and the second is the path to airports.txt containing airport information. The topology runs on a local Storm cluster for 10 seconds before shutting down.

FlightsDataReader Class: This class is a custom Storm spout that reads a JSON-formatted flight dataset from a file and emits each flight record as a tuple to the topology. In the open() method, it initializes the file reader using the path provided in the Storm configuration. The nextTuple() method reads the entire file once, parses the JSON, and iterates over the "states" array, which contains individual flight entries. For each flight with 17 fields, it emits a tuple with those fields (handling nulls as empty strings) using the SpoutOutputCollector. The declareOutputFields() method defines the schema of each emitted tuple (e.g., call sign, latitude, altitude, etc.). The class also implements ack(), fail(), and close() to handle basic message acknowledgement and cleanup.

HubIdentifier class: The HubIdentifier class is a Storm bolt that filters flights to identify those near major airports and emits the airport-city, airport-code, and call-sign for each. It loads airport data during setup and, for each flight, checks whether it is within 20 miles in latitude or longitude of any known airport. If a nearby airport is found, the flight is passed on for further processing.

AirlineSorter class: This class is a Storm bolt that groups and counts flights by airline code for each airport. It receives tuples containing the airport city, airport code, and flight call sign. For each tuple, it extracts the first three characters of the call sign as the airline code and updates a count of flights for that airline at the corresponding airport. In the cleanup() method, it prints out a summary of the total flights per airline at each airport, along with the total number of flights at that airport. *Counterpart Class:* This thread runs in the background and handles the opponent's actions during the game. It waits for input from the other player, marks their move on the board, and checks for a win or draw. If the game ends, it shows the result, and disables the board. It also flips the turn back to the player after each valid move.

A general data flow goes as this. The topology processes flight data by first reading a JSON-formatted flights.txt file through the FlightsDataReader spout, which emits tuples containing flight details. These are passed to the HubIdentifier bolt, which loads airport location data from airports.txt and filters flights to identify those within 20 miles (in latitude or longitude) of a listed airport. For each qualifying flight, it emits the nearest airport and the flight's call sign to the AirlineSorter bolt, which groups flights by airport and airline code, tracks counts, and prints a summary of airline activity per airport when the topology finishes.

Source Code:

TopologyMain Class:

```java
import backtype.storm.Config;
import backtype.storm.LocalCluster;
import backtype.storm.topology.TopologyBuilder;
import backtype.storm.tuple.Fields;

import spouts.FlightsDataReader;
import bolts.HubIdentifier;
import bolts.AirlineSorter;

public class TopologyMain {
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: TopologyMain <flights.txt path> <airports.txt path>");
            System.exit(1);
        }

        String flightsPath = args[0];
        String airportsPath = args[1];

        // Create topology
        TopologyBuilder builder = new TopologyBuilder();

        builder.setSpout("flights-reader", new FlightsDataReader());

        builder.setBolt("hub-identifier", new HubIdentifier())
                .shuffleGrouping("flights-reader");

        builder.setBolt("airline-sorter", new AirlineSorter(), 1)
                .fieldsGrouping("hub-identifier", new Fields("airport.city"));

        // Configuration
        Config conf = new Config();
        conf.put("FlightsFile", flightsPath);
        conf.put("AirportsData", airportsPath);
        conf.setDebug(false);
        conf.put(Config.TOPOLOGY_MAX_SPOUT_PENDING, 1);

        // Run locally
        LocalCluster cluster = new LocalCluster();
        cluster.submitTopology("Flight-Data-Analysis", conf, builder.createTopology());

        Thread.sleep(10000);
        cluster.shutdown();
    }
}
```

FlightsDataReader Class:

```java
package spouts;

import backtype.storm.spout.SpoutOutputCollector;
import backtype.storm.task.TopologyContext;
import backtype.storm.topology.OutputFieldsDeclarer;
import backtype.storm.topology.base.BaseRichSpout;
import backtype.storm.tuple.Fields;
import backtype.storm.tuple.Values;

import java.io.BufferedReader;
import java.io.FileReader;
import java.util.Map;
import org.json.JSONArray;
import org.json.JSONObject;


public class FlightsDataReader extends BaseRichSpout {
    private SpoutOutputCollector collector;
    private FileReader fileReader;
    private boolean completed = false;

    public void ack(Object msgId) {
        System.out.println("OK: " + msgId);
    }

    public void fail(Object msgId) {
        System.out.println("FAIL: " + msgId);
    }

    public void close() {}

    public void nextTuple() {
        if (completed) {
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {}
            return;
        }

        try {
            StringBuilder jsonBuilder = new StringBuilder();
            BufferedReader reader = new BufferedReader(fileReader);
            String line;
            while ((line = reader.readLine()) != null) {
                jsonBuilder.append(line);
            }

            JSONObject root = new JSONObject(jsonBuilder.toString());
            JSONArray states = root.getJSONArray("states");

            for (int i = 0; i < states.length(); i++) {
                JSONArray flight = states.getJSONArray(i);
                if (flight.length() == 17) {
                    Object[] fields = new Object[17];
                    for (int j = 0; j < 17; j++) {
                        fields[j] = (flight.isNull(j)) ? "" : flight.get(j).toString().trim();
                    }
                    // System.out.println("EMITTING: " + Arrays.toString(fields));
                    collector.emit(new Values(fields), fields[1]);
                }
            }

        } catch (Exception e) {
            e.printStackTrace();
            throw new RuntimeException("Error parsing flights.json", e);
        }

        completed = true;
    }



    public void open(Map conf, TopologyContext context, SpoutOutputCollector collector) {
        this.collector = collector;
        try {
            String filePath = conf.get("FlightsFile").toString();
            System.out.println("Reading flights data from: " + filePath);
            this.fileReader = new FileReader(filePath);
        } catch (Exception e) {
            throw new RuntimeException("Error reading flight file", e);
        }
    }

    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields(
            "transponder_address", "call_sign", "origin_country", "last_contact1", "last_contact2",
            "longitude", "latitude", "altitude_barometric", "on_ground", "velocity", "heading",
            "vertical_rate", "sensors", "altitude_geometric", "transponder_code", "special_purpose",
"origin"
        )); // Adjusted couple of names to make them more meaningful and easy to fetch
    }
}
```

HubIdentifier Class:

```java
package bolts;

import backtype.storm.topology.BasicOutputCollector;
import backtype.storm.topology.OutputFieldsDeclarer;
import backtype.storm.topology.base.BaseBasicBolt;
import backtype.storm.tuple.Fields;
import backtype.storm.tuple.Tuple;
import backtype.storm.tuple.Values;
import backtype.storm.task.TopologyContext;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

public class HubIdentifier extends BaseBasicBolt {

    private List<Airport> airports = new ArrayList<>();

    public void prepare(Map stormConf, TopologyContext context) { // Initialize the airports list
        String airportsFile = stormConf.get("AirportsData").toString();
        try (BufferedReader br = new BufferedReader(new FileReader(airportsFile))) {
            String line;
            while ((line = br.readLine()) != null) {
                // There are only 40 entries in the airports.txt file, So there was no need to select top 
40 entries.
                String[] parts = line.split(",");
                if (parts.length >= 4) {
                    String city = parts[0].trim();
                    String code = parts[1].trim();
                    double lat = Double.parseDouble(parts[2].trim());
                    double lon = Double.parseDouble(parts[3].trim());
                    airports.add(new Airport(city, code, lat, lon));
                }
            }
        } catch (Exception e) {
            throw new RuntimeException("Error reading airports file", e);
        }
    }

    public void execute(Tuple input, BasicOutputCollector collector) {
        try {
            String callSign = input.getStringByField("call_sign").trim();
            double flightLat = Double.parseDouble(input.getStringByField("latitude"));
            double flightLon = Double.parseDouble(input.getStringByField("longitude"));
            double verticalRate = Double.parseDouble(input.getStringByField("vertical_rate"));
            double velocity = Double.parseDouble(input.getStringByField("velocity"));
            double altitude = Double.parseDouble(input.getStringByField("altitude_geometric"));
            boolean onGround = Boolean.parseBoolean(input.getStringByField("on_ground"));

            if (callSign == null || callSign.trim().isEmpty()) {
                return; // Skip flights with no call sign
            }

            // Filter out flyovers
            if (!onGround && verticalRate == 0) {
                return; // Skip flights that are not on the ground and have no vertical rate
            }

            if (velocity > 200) {
                return; // Skip high-speed flights
            }

            if (altitude > 1000) {
                return; // Skip high altitude flights
            }


            Airport nearest = null;
            double minDistance = Double.MAX_VALUE;

            double latThreshold = 20.0 / 70.0;   //  0.2857 According to the given proximity rule
            double lonThreshold = 20.0 / 45.0;   // 0.4444 According to the given proximity rule

            for (Airport airport : airports) {
                boolean nearLat = Math.abs(flightLat - airport.latitude) <= latThreshold;
                boolean nearLon = Math.abs(flightLon - airport.longitude) <= lonThreshold;

                if (nearLat || nearLon) { // Near either latitude or longitude
                    // System.out.println("NEAR " + airport.code + ": " + callSign);
                    double dist = Math.sqrt(Math.pow(flightLat - airport.latitude, 2) + Math.pow(flightLon 
- airport.longitude, 2)); //calculating the Euclidean distance
                    if (nearest == null || dist < minDistance) {
                        nearest = airport;
                        minDistance = dist;
                    }
                }
            }

            if (nearest != null) {
                System.out.println("NEAREST " + nearest.city +" (" + nearest.code + "): " + callSign);
                // Airline code is typically the first 3 letters of the call sign
                String airlineCode = callSign.length() >= 3 ? callSign.substring(0, 3).trim() : callSign;
                collector.emit(new Values(nearest.city, nearest.code, airlineCode));
            }
        } catch (Exception ignored) {}
    }

    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("airport.city", "airport.code", "call_sign"));
    }

    public void cleanup() {}


    private static class Airport implements Serializable {
        String city, code;
        double latitude, longitude;

        Airport(String city, String code, double latitude, double longitude) {
            this.city = city;
            this.code = code;
            this.latitude = latitude;
            this.longitude = longitude;
        }
    }
}
```

AirlineSorter Class:

```
package bolts;

import backtype.storm.task.TopologyContext;
import backtype.storm.topology.BasicOutputCollector;
import backtype.storm.topology.OutputFieldsDeclarer;
import backtype.storm.topology.base.BaseBasicBolt;
import backtype.storm.tuple.Tuple;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class AirlineSorter extends BaseBasicBolt {

    // This is used to store the number of flights for each airline at each airport
    private Map<String, Map<String, Integer>> stats;

    // Map from airport code to city name
    private Map<String, String> codeToCity;
```

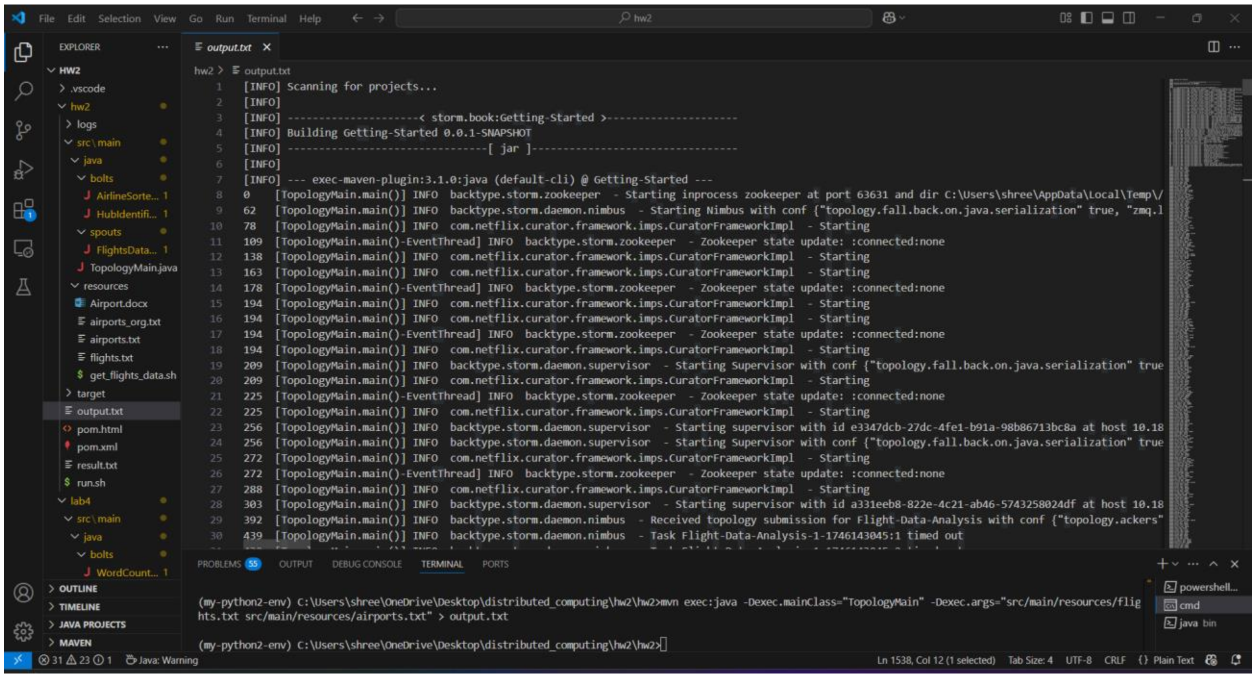
```java
    @Override
    public void prepare(Map stormConf, TopologyContext context) {
        stats = new HashMap<>();
        codeToCity = new HashMap<>();
    }

    @Override
    public void execute(Tuple input, BasicOutputCollector collector) {
        String airportCity = input.getStringByField("airport.city");
        String airportCode = input.getStringByField("airport.code");
        String airlineCode = input.getStringByField("call_sign");

        // Store code -> city mapping
        codeToCity.putIfAbsent(airportCode, airportCity);

        // Keyed by airport code
        stats.putIfAbsent(airportCode, new HashMap<String,Integer>());
        Map<String, Integer> airlineMap = stats.get(airportCode);
        airlineMap.put(airlineCode, airlineMap.getOrDefault(airlineCode, 0) + 1);
    }

    @Override
    public void cleanup() {
        System.out.println("-- Flight Counter --");

        for (Map.Entry<String, Map<String, Integer>> airportEntry : stats.entrySet()) {
            String airport = airportEntry.getKey();
            Map<String, Integer> airlineMap = airportEntry.getValue();
            String cityName = codeToCity.getOrDefault(airport, "Unknown");

            System.out.println("At Airport: " + airport + " (" + cityName + ")");
            int total = 0;

            List<Map.Entry<String, Integer>> sortedEntries = new ArrayList<>(airlineMap.entrySet());
            sortedEntries.sort((a, b) -> b.getValue().compareTo(a.getValue()));

            for (Map.Entry<String, Integer> entry : sortedEntries) {
                System.out.println("\t" + entry.getKey() + ": " + entry.getValue());
                total += entry.getValue();
            }

            System.out.println("\ttotal #flights = " + total);
            System.out.println();
        }
    }



    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        // No output fields from final bolt
    }
}
```

Execution Outputs:

Two scenarios were given to test with the implementation. Below are the outputs for each.

Single-threaded execution:

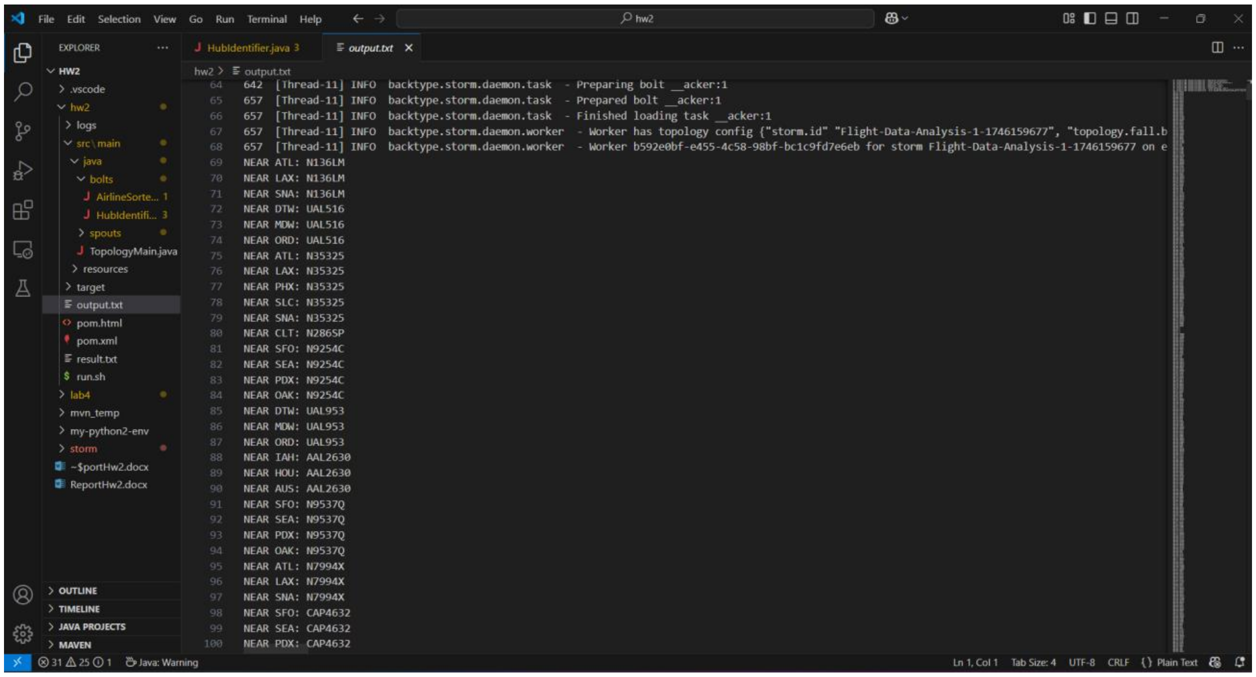
Execution start.



```
[INFO] Scanning for projects...
[INFO]
[INFO] --------------------< storm.book:Getting-Started >--------------------
[INFO] Building Getting-Started 0.0.1-SNAPSHOT
[INFO] -------------------------------[ jar ]--------------------------------
[INFO]
[INFO] --- exec-maven-plugin:3.1.0:java (default-cli) @ Getting-Started ---
0    [TopologyMain.main()] INFO  backtype.storm.zookeeper  - Starting inprocess zookeeper at port 63631 and dir C:\Users\shree\AppData\Local\Temp\/
62   [TopologyMain.main()] INFO  backtype.storm.daemon.nimbus  - Starting Nimbus with conf {"topology.fall.back.on.java.serialization" true, "zmq.l
78   [TopologyMain.main()] INFO  com.netflix.curator.framework.imps.CuratorFrameworkImpl  - Starting
109  [TopologyMain.main()-EventThread] INFO  backtype.storm.zookeeper  - Zookeeper state update: :connected:none
138  [TopologyMain.main()] INFO  com.netflix.curator.framework.imps.CuratorFrameworkImpl  - Starting
163  [TopologyMain.main()] INFO  com.netflix.curator.framework.imps.CuratorFrameworkImpl  - Starting
178  [TopologyMain.main()-EventThread] INFO  backtype.storm.zookeeper  - Zookeeper state update: :connected:none
194  [TopologyMain.main()] INFO  com.netflix.curator.framework.imps.CuratorFrameworkImpl  - Starting
194  [TopologyMain.main()] INFO  com.netflix.curator.framework.imps.CuratorFrameworkImpl  - Starting
194  [TopologyMain.main()-EventThread] INFO  backtype.storm.zookeeper  - Zookeeper state update: :connected:none
194  [TopologyMain.main()] INFO  com.netflix.curator.framework.imps.CuratorFrameworkImpl  - Starting
209  [TopologyMain.main()] INFO  backtype.storm.daemon.supervisor  - Starting Supervisor with conf {"topology.fall.back.on.java.serialization" true
209  [TopologyMain.main()] INFO  com.netflix.curator.framework.imps.CuratorFrameworkImpl  - Starting
225  [TopologyMain.main()-EventThread] INFO  backtype.storm.zookeeper  - Zookeeper state update: :connected:none
225  [TopologyMain.main()] INFO  com.netflix.curator.framework.imps.CuratorFrameworkImpl  - Starting
256  [TopologyMain.main()] INFO  backtype.storm.daemon.supervisor  - Starting supervisor with id e3347dcb-27dc-4fe1-b91a-98b86713bc8a at host 10.18
256  [TopologyMain.main()] INFO  backtype.storm.daemon.supervisor  - Starting Supervisor with conf {"topology.fall.back.on.java.serialization" true
272  [TopologyMain.main()] INFO  com.netflix.curator.framework.imps.CuratorFrameworkImpl  - Starting
272  [TopologyMain.main()-EventThread] INFO  backtype.storm.zookeeper  - Zookeeper state update: :connected:none
288  [TopologyMain.main()] INFO  com.netflix.curator.framework.imps.CuratorFrameworkImpl  - Starting
303  [TopologyMain.main()] INFO  backtype.storm.daemon.supervisor  - Starting supervisor with id a331eeb8-822e-4c21-ab46-5743258024df at host 10.18
392  [TopologyMain.main()] INFO  backtype.storm.daemon.nimbus  - Received topology submission for Flight-Data-Analysis with conf {"topology.ackers"
439  [TopologyMain.main()] INFO  backtype.storm.daemon.nimbus  - Task Flight-Data-Analysis-1-1746143045:1 timed out
```

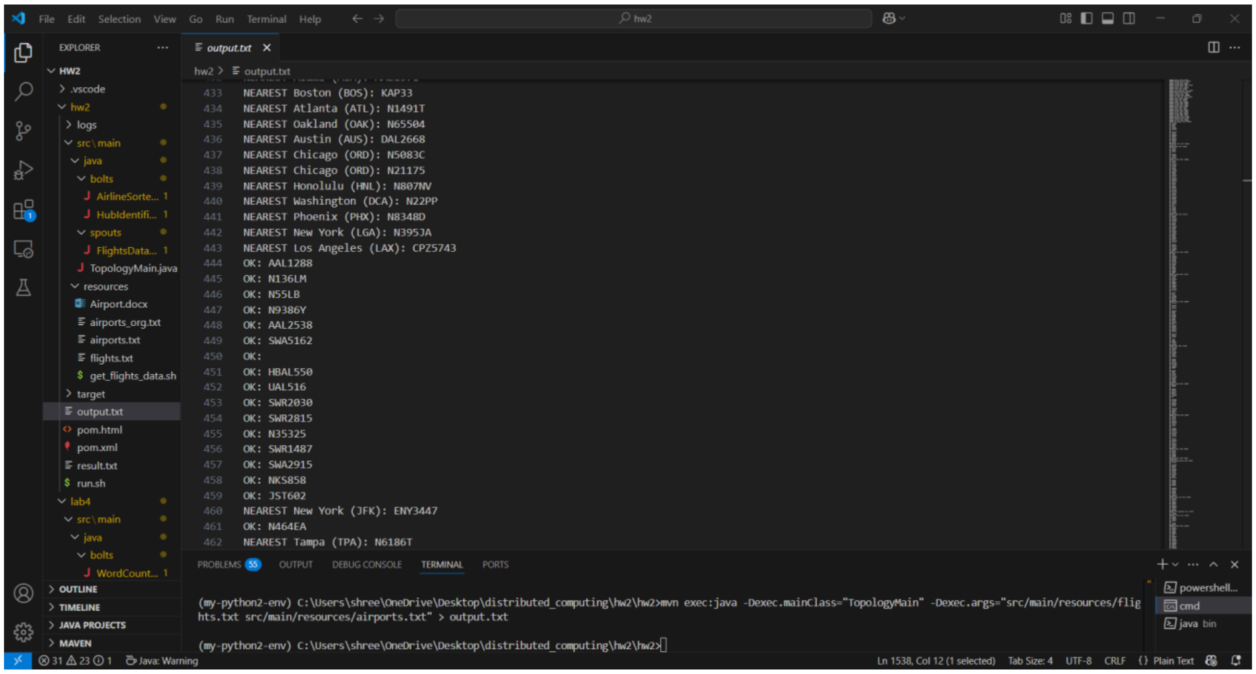
```
(my-python2-env) C:\Users\shree\OneDrive\Desktop\distributed_computing\hw2\hw2>mvn exec:java -Dexec.mainClass="TopologyMain" -Dexec.args="src/main/resources/flig
hts.txt src/main/resources/airports.txt" > output.txt

(my-python2-env) C:\Users\shree\OneDrive\Desktop\distributed_computing\hw2\hw2>
```

One flight mapping multiple airports. Original implementation.



```
642  [Thread-11] INFO  backtype.storm.daemon.task  - Preparing bolt __acker:1
657  [Thread-11] INFO  backtype.storm.daemon.task  - Prepared bolt __acker:1
657  [Thread-11] INFO  backtype.storm.daemon.task  - Finished loading task __acker:1
657  [Thread-11] INFO  backtype.storm.daemon.worker  - Worker has topology config {"storm.id" "Flight-Data-Analysis-1-1746159677", "topology.fall.b
657  [Thread-11] INFO  backtype.storm.daemon.worker  - Worker b592e0bf-e455-4c58-98bf-bc1c9fd7e6eb for storm Flight-Data-Analysis-1-1746159677 on e
NEAR ATL: N136LM
NEAR LAX: N136LM
NEAR SNA: N136LM
NEAR DTW: UAL516
NEAR MDW: UAL516
NEAR ORD: UAL516
NEAR ATL: N35325
NEAR LAX: N35325
NEAR PHX: N35325
NEAR SLC: N35325
NEAR SNA: N35325
NEAR CLT: N286SP
NEAR SFO: N9254C
NEAR SEA: N9254C
NEAR PDX: N9254C
NEAR OAK: N9254C
NEAR DTW: UAL953
NEAR MDW: UAL953
NEAR ORD: UAL953
NEAR IAH: AAL2630
NEAR HOU: AAL2630
NEAR AUS: AAL2630
NEAR SFO: N9537Q
NEAR SEA: N9537Q
NEAR PDX: N9537Q
NEAR OAK: N9537Q
NEAR ATL: N7994X
NEAR LAX: N7994X
NEAR SNA: N7994X
NEAR SFO: CAP4632
NEAR SEA: CAP4632
NEAR PDX: CAP4632
```

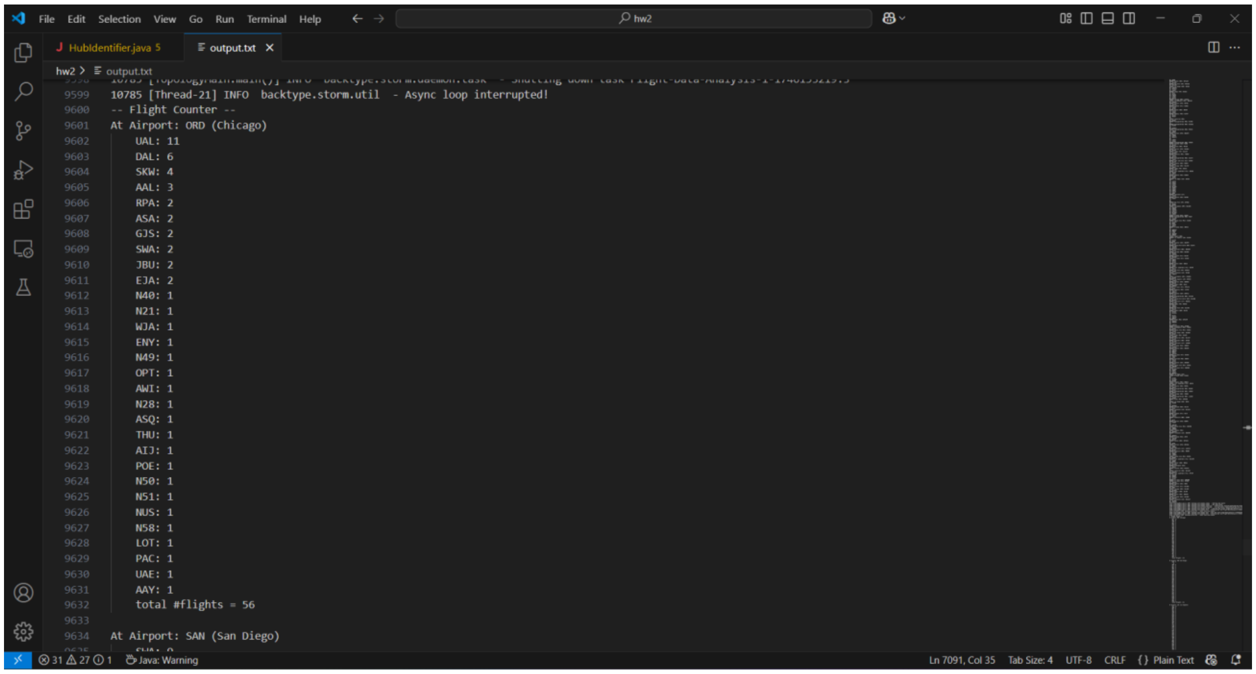
Receiving acknowledgement



```
NEAREST Boston (BOS): KAP33
NEAREST Atlanta (ATL): N1491T
NEAREST Oakland (OAK): N65504
NEAREST Austin (AUS): DAL2668
NEAREST Chicago (ORD): N5083C
NEAREST Chicago (ORD): N21175
NEAREST Honolulu (HNL): N807NV
NEAREST Washington (DCA): N22PP
NEAREST Phoenix (PHX): N8348D
NEAREST New York (LGA): N395JA
NEAREST Los Angeles (LAX): CPZ5743
OK: AAL1288
OK: N136LM
OK: N55LB
OK: N9386Y
OK: AAL2538
OK: SWA5162
OK:
OK: HBAL550
OK: UAL516
OK: SWR2030
OK: SWR2815
OK: N35325
OK: SWR1487
OK: SWA2915
OK: NKS858
OK: JST602
NEAREST New York (JFK): ENY3447
OK: N464EA
NEAREST Tampa (TPA): N6186T
```

```
(my-python2-env) C:\Users\shree\OneDrive\Desktop\distributed_computing\hw2\hw2>mvn exec:java -Dexec.mainClass="TopologyMain" -Dexec.args="src/main/resources/flig
hts.txt src/main/resources/airports.txt" > output.txt

(my-python2-env) C:\Users\shree\OneDrive\Desktop\distributed_computing\hw2\hw2>
```

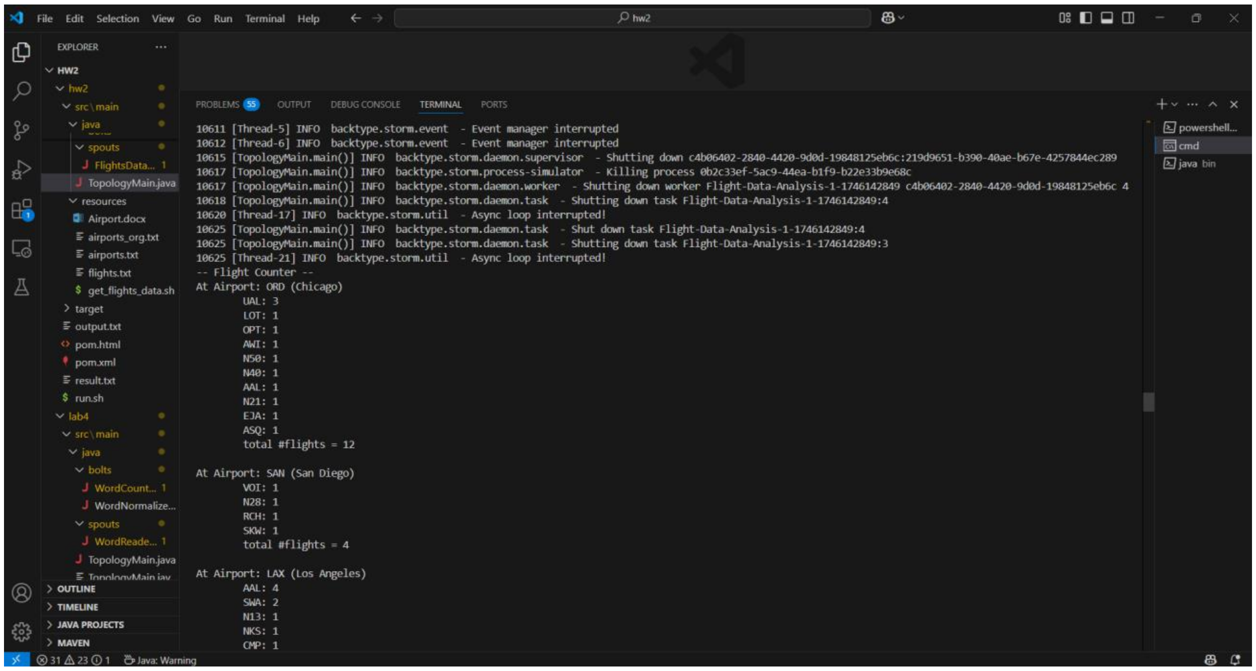
Original Implementation- Mapping near airport.



```
10785 [Thread-21] INFO  backtype.storm.util  - Async loop interrupted!
-- Flight Counter --
At Airport: ORD (Chicago)
    UAL: 11
    DAL: 6
    SKW: 4
    AAL: 3
    RPA: 2
    ASA: 2
    GJS: 2
    SWA: 2
    JBU: 2
    EJA: 2
    N40: 1
    N21: 1
    WJA: 1
    ENY: 1
    N49: 1
    OPT: 1
    AWI: 1
    N28: 1
    ASQ: 1
    THU: 1
    AIJ: 1
    POE: 1
    N50: 1
    N51: 1
    NUS: 1
    N58: 1
    LOT: 1
    PAC: 1
    UAE: 1
    AAY: 1
    total #flights = 56

At Airport: SAN (San Diego)
```

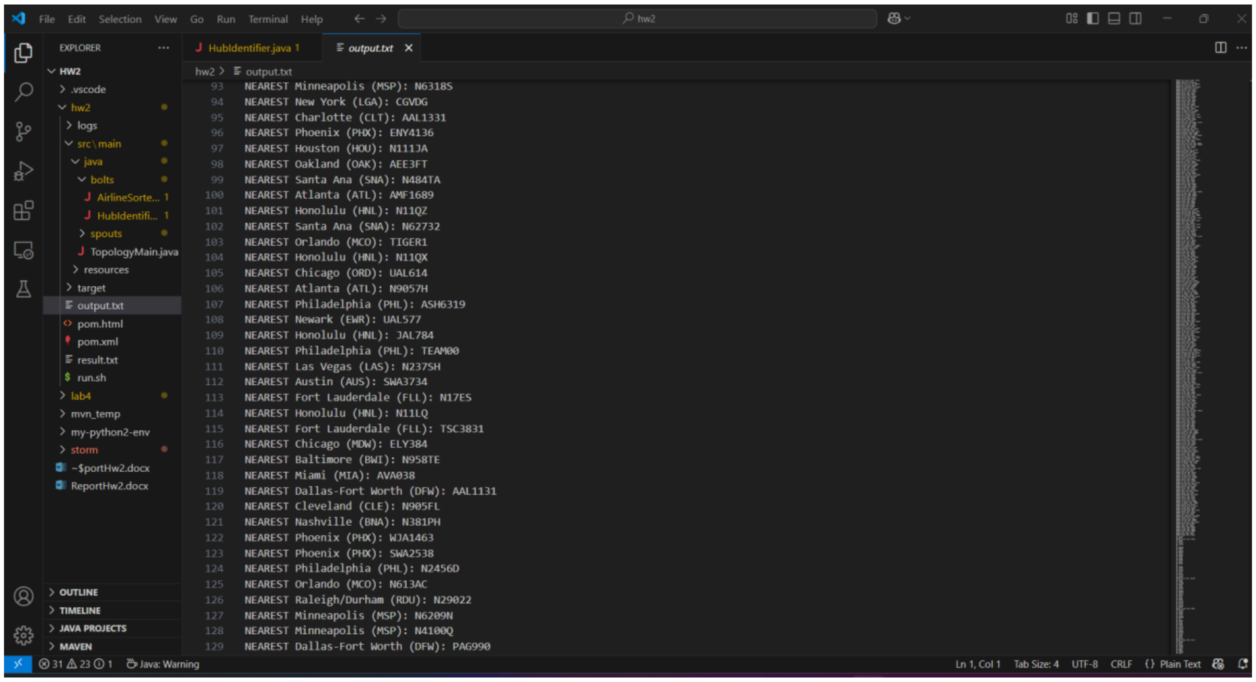
Additional feature(s):

Sorted and filtered output.



```
10611 [Thread-5] INFO  backtype.storm.event  - Event manager interrupted
10612 [Thread-6] INFO  backtype.storm.event  - Event manager interrupted
10615 [TopologyMain.main()] INFO  backtype.storm.daemon.supervisor  - Shutting down c4b06402-2840-4420-9d0d-19848125eb6c:219d9651-b390-40ae-b67e-4257844ec289
10617 [TopologyMain.main()] INFO  backtype.storm.process-simulator  - Killing process 0b2c33ef-5ac9-44ea-b1f9-b22e33b9e68c
10617 [TopologyMain.main()] INFO  backtype.storm.daemon.worker  - Shutting down worker Flight-Data-Analysis-1-1746142849 c4b06402-2840-4420-9d0d-19848125eb6c 4
10618 [TopologyMain.main()] INFO  backtype.storm.daemon.task  - Shutting down task Flight-Data-Analysis-1-1746142849:4
10620 [Thread-17] INFO  backtype.storm.util  - Async loop interrupted!
10625 [TopologyMain.main()] INFO  backtype.storm.daemon.task  - Shut down task Flight-Data-Analysis-1-1746142849:4
10625 [TopologyMain.main()] INFO  backtype.storm.daemon.task  - Shutting down task Flight-Data-Analysis-1-1746142849:3
10625 [Thread-21] INFO  backtype.storm.util  - Async loop interrupted!
-- Flight Counter --
At Airport: ORD (Chicago)
        UAL: 3
        LOT: 1
        OPT: 1
        AWI: 1
        N50: 1
        N40: 1
        AAL: 1
        N21: 1
        EJA: 1
        ASQ: 1
        total #flights = 12

At Airport: SAN (San Diego)
        VOI: 1
        N28: 1
        RCH: 1
        SKW: 1
        total #flights = 4

At Airport: LAX (Los Angeles)
        AAL: 4
        SWA: 2
        N13: 1
        NKS: 1
        CMP: 1
```

Mapping to only one nearest airport



```
NEAREST Minneapolis (MSP): N6318S
NEAREST New York (LGA): CGVDG
NEAREST Charlotte (CLT): AAL1331
NEAREST Phoenix (PHX): ENY4136
NEAREST Houston (HOU): N111JA
NEAREST Oakland (OAK): AEE3FT
NEAREST Santa Ana (SNA): N484TA
NEAREST Atlanta (ATL): AMF1689
NEAREST Honolulu (HNL): N11QZ
NEAREST Santa Ana (SNA): N62732
NEAREST Orlando (MCO): TIGER1
NEAREST Honolulu (HNL): N11QX
NEAREST Chicago (ORD): UAL614
NEAREST Atlanta (ATL): N9057H
NEAREST Philadelphia (PHL): ASH6319
NEAREST Newark (EWR): UAL577
NEAREST Honolulu (HNL): JAL784
NEAREST Philadelphia (PHL): TEAM00
NEAREST Las Vegas (LAS): N237SH
NEAREST Austin (AUS): SWA3734
NEAREST Fort Lauderdale (FLL): N17ES
NEAREST Honolulu (HNL): N11LQ
NEAREST Fort Lauderdale (FLL): TSC3831
NEAREST Chicago (MDW): ELY384
NEAREST Baltimore (BWI): N958TE
NEAREST Miami (MIA): AVA038
NEAREST Dallas-Fort Worth (DFW): AAL1131
NEAREST Cleveland (CLE): N905FL
NEAREST Nashville (BNA): N381PH
NEAREST Phoenix (PHX): WJA1463
NEAREST Phoenix (PHX): SWA2538
NEAREST Philadelphia (PHL): N2456D
NEAREST Orlando (MCO): N613AC
NEAREST Raleigh/Durham (RDU): N29022
NEAREST Minneapolis (MSP): N6209N
NEAREST Minneapolis (MSP): N4100Q
NEAREST Dallas-Fort Worth (DFW): PAG990
```

Discussions:

Additional Features:

1. Sorting the airline based on the number of flights it has in an airport.
2. More accurate flight information:
    a. Filtered the airplanes according to different parameters such as altitude, vertical rate.
    b. Improved the airplanes mappings to its nearest airport

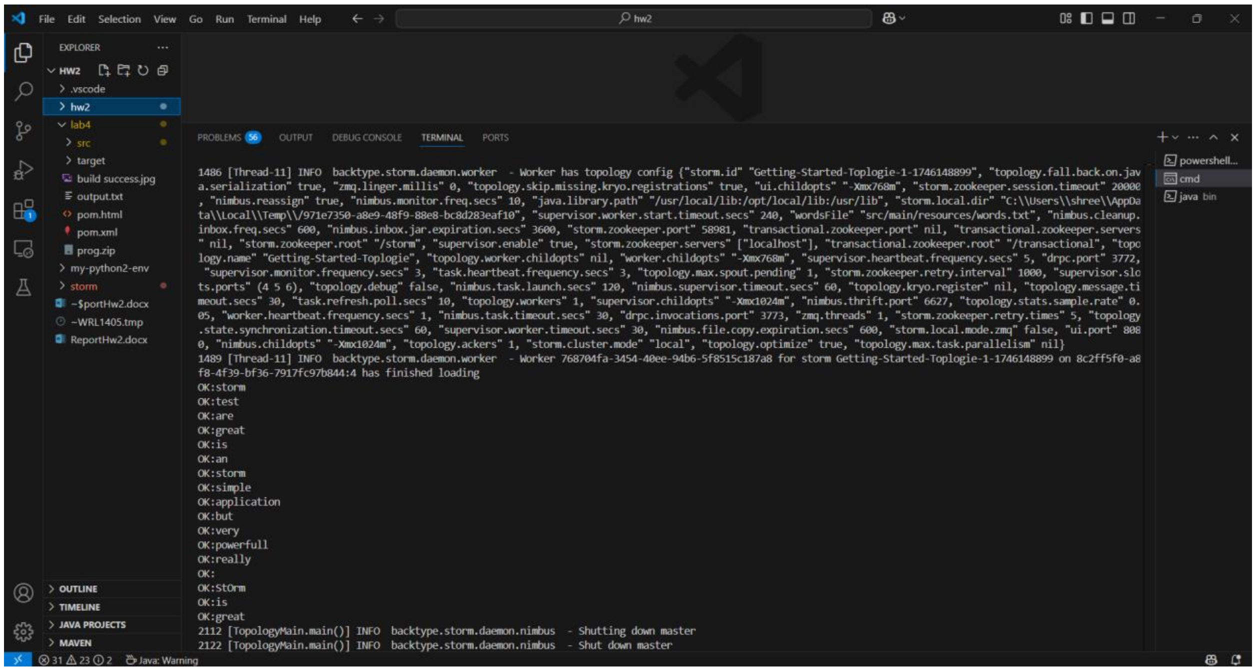
To implement the feature of sorting airlines based on how many flights they have at each airport, the AirlineSorter class uses a nested map structure called stats. This map keeps track of flight counts, with the airport code as the outer key and a map of airline codes and their respective counts as the inner values. The airline code is extracted as the first three characters of the call sign, which is passed from the HubIdentifier class. As flight data comes in during the execute() method, the corresponding airline's count is updated. Then, in the cleanup() method, each airport's airline data is collected, turned into a list, and sorted in descending order based on flight count. Finally, the class prints a summary for each airport showing its city, the airlines that operated there, and how many flights each had.

For the Implementation of more accurate flight data. The following implementations are done.

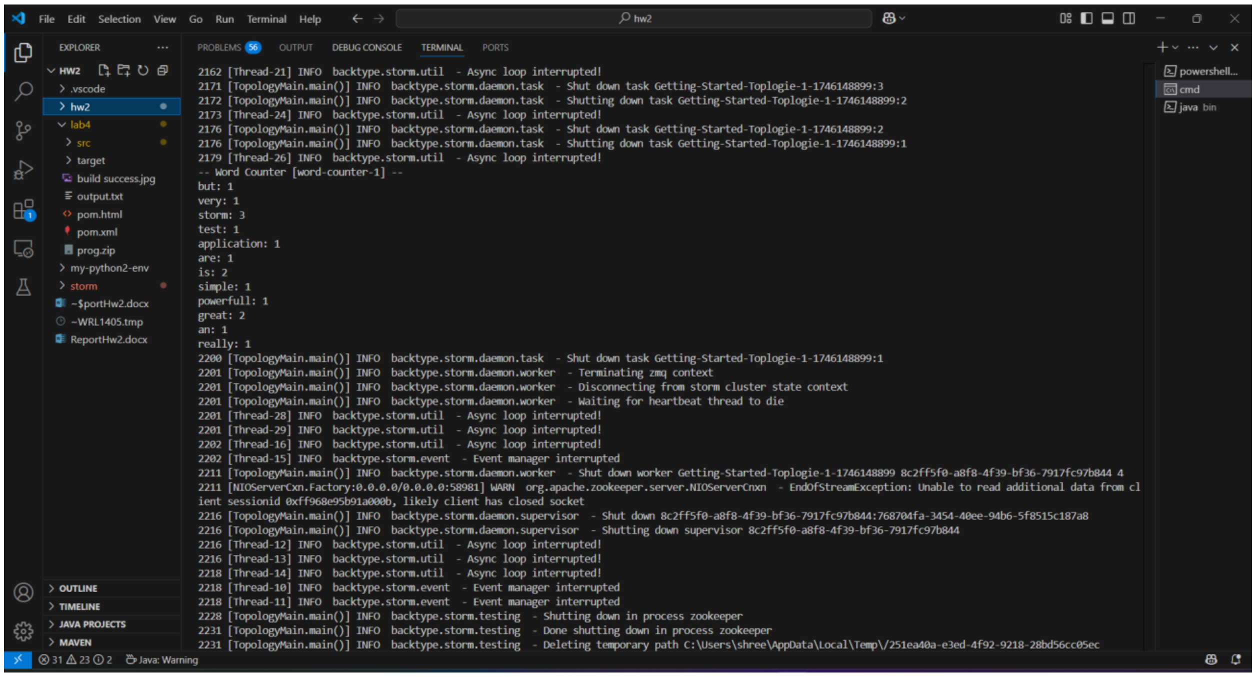
In the HubIdentifier class, a set of filters is used to focus on flights that are actually relating with an airport, such as those taking off, landing, or on the ground, rather than those simply flying overhead. The class first checks if the flight has a valid call sign, as missing call signs often indicate incomplete or less useful data. It then filters out flights that are airborne but not changing altitude, which usually means they are just cruising by. Flights moving faster than 200 meters per second are also excluded, since they are likely passing over the area at high speed. Additionally, flights above 1000 feet are not considered, as the focus is on aircraft closer to the ground or near the airport. Once these filters are applied, the remaining flights are checked for proximity to known airports based on whether they are within about 20 miles in either latitude or longitude. This approach ensures that only flights that are related or connected to an airport are included in the topology.

In the initial sample output, it was observed that some flights were being mapped to multiple airports due to the broad proximity rule, which considers a flight near any airport if it is within 20 miles in either latitude or longitude. While this approach captures potential proximity, it can also introduce ambiguity and inflate the data, since a single flight might be linked to several airports. To avoid this confusion, the HubIdentifier class was designed to identify only the nearest airport for each flight. Instead of emitting a flight for every matching airport within the proximity range, the implementation keeps track of the closest one by calculating the Euclidean distance between the flight's coordinates and each airport. This is done by looping through all airports and updating the minimum distance and corresponding airport whenever a closer one is found. As a result, each flight is uniquely and more accurately associated with just one airport, which simplifies the analysis and leads to more meaningful flight-to-airport mappings.

Lab 4:



```
1486 [Thread-11] INFO  backtype.storm.daemon.worker  - Worker has topology config {"storm.id" "Getting-Started-Toplogie-1-1746148899", "topology.fall.back.on.jav
a.serialization" true, "zmq.linger.millis" 0, "topology.skip.missing.kryo.registrations" true, "ui.childopts" "-Xmx768m", "storm.zookeeper.session.timeout" 20000
, "nimbus.reassign" true, "nimbus.monitor.freq.secs" 10, "java.library.path" "/usr/local/lib:/opt/local/lib:/usr/lib", "storm.local.dir" "C:\\Users\\shree\\AppDa
ta\\Local\\Temp\\/971e7350-a8e9-48f9-88e8-bc8d283eaf10", "supervisor.worker.start.timeout.secs" 240, "wordsFile" "src/main/resources/words.txt", "nimbus.cleanup.
inbox.freq.secs" 600, "nimbus.inbox.jar.expiration.secs" 3600, "storm.zookeeper.port" 58981, "transactional.zookeeper.port" nil, "transactional.zookeeper.servers
" nil, "storm.zookeeper.root" "/storm", "supervisor.enable" true, "storm.zookeeper.servers" ["localhost"], "transactional.zookeeper.root" "/transactional", "topo
logy.name" "Getting-Started-Toplogie", "topology.worker.childopts" nil, "worker.childopts" "-Xmx768m", "supervisor.heartbeat.frequency.secs" 5, "drpc.port" 3772,
 "supervisor.monitor.frequency.secs" 3, "task.heartbeat.frequency.secs" 3, "topology.max.spout.pending" 1, "storm.zookeeper.retry.interval" 1000, "supervisor.slo
ts.ports" (4 5 6), "topology.debug" false, "nimbus.task.launch.secs" 120, "nimbus.supervisor.timeout.secs" 60, "topology.kryo.register" nil, "topology.message.ti
meout.secs" 30, "task.refresh.poll.secs" 10, "topology.workers" 1, "supervisor.childopts" "-Xmx1024m", "nimbus.thrift.port" 6627, "topology.stats.sample.rate" 0.
05, "worker.heartbeat.frequency.secs" 1, "nimbus.task.timeout.secs" 30, "drpc.invocations.port" 3773, "zmq.threads" 1, "storm.zookeeper.retry.times" 5, "topology
.state.synchronization.timeout.secs" 60, "supervisor.worker.timeout.secs" 30, "nimbus.file.copy.expiration.secs" 600, "storm.local.mode.zmq" false, "ui.port" 808
0, "nimbus.childopts" "-Xmx1024m", "topology.ackers" 1, "storm.cluster.mode" "local", "topology.optimize" true, "topology.max.task.parallelism" nil}
1489 [Thread-11] INFO  backtype.storm.daemon.worker  - Worker 768704fa-3454-40ee-94b6-5f8515c187a8 for storm Getting-Started-Toplogie-1-1746148899 on 8c2ff5f0-a8
f8-4f39-bf36-7917fc97b844:4 has finished loading
OK:storm
OK:test
OK:are
OK:great
OK:is
OK:an
OK:storm
OK:simple
OK:application
OK:but
OK:very
OK:powerfull
OK:really
OK:
OK:StOrm
OK:is
OK:great
2112 [TopologyMain.main()] INFO  backtype.storm.daemon.nimbus  - Shutting down master
2122 [TopologyMain.main()] INFO  backtype.storm.daemon.nimbus  - Shut down master
```



```
2162 [Thread-21] INFO  backtype.storm.util  - Async loop interrupted!
2171 [TopologyMain.main()] INFO  backtype.storm.daemon.task  - Shut down task Getting-Started-Toplogie-1-1746148899:3
2172 [TopologyMain.main()] INFO  backtype.storm.daemon.task  - Shutting down task Getting-Started-Toplogie-1-1746148899:2
2173 [Thread-24] INFO  backtype.storm.util  - Async loop interrupted!
2176 [TopologyMain.main()] INFO  backtype.storm.daemon.task  - Shut down task Getting-Started-Toplogie-1-1746148899:2
2176 [TopologyMain.main()] INFO  backtype.storm.daemon.task  - Shutting down task Getting-Started-Toplogie-1-1746148899:1
2179 [Thread-26] INFO  backtype.storm.util  - Async loop interrupted!
-- Word Counter [word-counter-1] --
but: 1
very: 1
storm: 3
test: 1
application: 1
are: 1
is: 2
simple: 1
powerfull: 1
great: 2
an: 1
really: 1
2200 [TopologyMain.main()] INFO  backtype.storm.daemon.task  - Shut down task Getting-Started-Toplogie-1-1746148899:1
2201 [TopologyMain.main()] INFO  backtype.storm.daemon.worker  - Terminating zmq context
2201 [TopologyMain.main()] INFO  backtype.storm.daemon.worker  - Disconnecting from storm cluster state context
2201 [TopologyMain.main()] INFO  backtype.storm.daemon.worker  - Waiting for heartbeat thread to die
2201 [Thread-28] INFO  backtype.storm.util  - Async loop interrupted!
2201 [Thread-29] INFO  backtype.storm.util  - Async loop interrupted!
2202 [Thread-16] INFO  backtype.storm.util  - Async loop interrupted!
2202 [Thread-15] INFO  backtype.storm.event  - Event manager interrupted
2211 [TopologyMain.main()] INFO  backtype.storm.daemon.worker  - Shut down worker Getting-Started-Toplogie-1-1746148899 8c2ff5f0-a8f8-4f39-bf36-7917fc97b844 4
2211 [NIOServerCxn.Factory:0.0.0.0/0.0.0.0:58981] WARN  org.apache.zookeeper.server.NIOServerCnxn  - EndOfStreamException: Unable to read additional data from cl
ient sessionid 0xff968e95b91a000b, likely client has closed socket
2216 [TopologyMain.main()] INFO  backtype.storm.daemon.supervisor  - Shut down 8c2ff5f0-a8f8-4f39-bf36-7917fc97b844:768704fa-3454-40ee-94b6-5f8515c187a8
2216 [TopologyMain.main()] INFO  backtype.storm.daemon.supervisor  - Shutting down supervisor 8c2ff5f0-a8f8-4f39-bf36-7917fc97b844
2216 [Thread-12] INFO  backtype.storm.util  - Async loop interrupted!
2216 [Thread-13] INFO  backtype.storm.util  - Async loop interrupted!
2218 [Thread-14] INFO  backtype.storm.util  - Async loop interrupted!
2218 [Thread-10] INFO  backtype.storm.event  - Event manager interrupted
2218 [Thread-11] INFO  backtype.storm.event  - Event manager interrupted
2228 [TopologyMain.main()] INFO  backtype.storm.testing  - Shutting down in process zookeeper
2231 [TopologyMain.main()] INFO  backtype.storm.testing  - Done shutting down in process zookeeper
2231 [TopologyMain.main()] INFO  backtype.storm.testing  - Deleting temporary path C:\Users\shree\AppData\Local\Temp\/251ea40a-e3ed-4f92-9218-28bd56cc05ec
```