AlgoExpert   Quad Layout   Java   12px   Sublime   Monokai   00:00:00

Prompt   Scratchpad   Our Solution(s)   Video Explanation   Run Code

Solution 1

```java
// Copyright © 2020 AlgoExpert, LLC. All rights reserved.

import java.util.*;

class Program {

  // O(a * (a + r) + a + r + alog(a)) time | O(a + r) space - where a is the number of airports and
  // r is the number of routes
  public static int airportConnections(
      List<String> airports, List<List<String>> routes, String startingAirport) {
    Map<String, AirportNode> airportGraph = createAirportGraph(airports, routes);
    List<AirportNode> unreachableAirportNodes =
        getUnreachableAirportNodes(airportGraph, airports, startingAirport);
    markUnreachableConnections(airportGraph, unreachableAirportNodes);
    return getMinNumberOfNewConnections(airportGraph, unreachableAirportNodes);
  }

  // O(a + r) time | O(a + r) space
  public static Map<String, AirportNode> createAirportGraph(
      List<String> airports, List<List<String>> routes) {
    Map<String, AirportNode> airportGraph = new HashMap<String, AirportNode>();
    for (String airport : airports) {
      airportGraph.put(airport, new AirportNode(airport));
    }
    for (List<String> route : routes) {
      String airport = route.get(0);
      String connection = route.get(1);
      airportGraph.get(airport).connections.add(connection);
    }
    return airportGraph;
  }

  // O(a + r) time | O(a) space
  public static List<AirportNode> getUnreachableAirportNodes(
      Map<String, AirportNode> airportGraph, List<String> airports, String startingAirport) {
    Set<String> visitedAirports = new HashSet<String>();
    depthFirstTraverseAirports(airportGraph, startingAirport, visitedAirports);

    List<AirportNode> unreachableAirportNodes = new ArrayList<AirportNode>();
    for (String airport : airports) {
      if (visitedAirports.contains(airport)) continue;
      AirportNode airportNode = airportGraph.get(airport);
      airportNode.isReachable = false;
      unreachableAirportNodes.add(airportNode);
    }
    return unreachableAirportNodes;
  }

  public static void depthFirstTraverseAirports(
      Map<String, AirportNode> airportGraph, String airport, Set<String> visitedAirports) {
    if (visitedAirports.contains(airport)) return;
    visitedAirports.add(airport);
    List<String> connections = airportGraph.get(airport).connections;
    for (String connection : connections) {
      depthFirstTraverseAirports(airportGraph, connection, visitedAirports);
    }
  }

  // O(a * (a + r)) time | O(a) space
  public static void markUnreachableConnections(
      Map<String, AirportNode> airportGraph, List<AirportNode> unreachableAirportNodes) {
    for (AirportNode airportNode : unreachableAirportNodes) {
      String airport = airportNode.airport;
      List<String> unreachableConnections = new ArrayList<String>();
      Set<String> visitedAirports = new HashSet<String>();
      depthFirstAddUnreachableConnections(
          airportGraph, airport, unreachableConnections, visitedAirports);
      airportNode.unreachableConnections = unreachableConnections;
    }
  }

  public static void depthFirstAddUnreachableConnections(
      Map<String, AirportNode> airportGraph,
      String airport,
      List<String> unreachableConnections,
      Set<String> visitedAirports) {
    if (airportGraph.get(airport).isReachable) return;
    if (visitedAirports.contains(airport)) return;
    visitedAirports.add(airport);
    unreachableConnections.add(airport);
    List<String> connections = airportGraph.get(airport).connections;
    for (String connection : connections) {
      depthFirstAddUnreachableConnections(
          airportGraph, connection, unreachableConnections, visitedAirports);
    }
  }

  // O(alog(a) + a + r) time | O(1) space
  public static int getMinNumberOfNewConnections(
      Map<String, AirportNode> airportGraph, List<AirportNode> unreachableAirportNodes) {
    unreachableAirportNodes.sort(
        (a1, a2) -> a2.unreachableConnections.size() - a1.unreachableConnections.size());
    int numberOfNewConnections = 0;
    for (AirportNode airportNode : unreachableAirportNodes) {
      if (airportNode.isReachable) continue;
      numberOfNewConnections++;
      for (String connection : airportNode.unreachableConnections) {
        airportGraph.get(connection).isReachable = true;
      }
    }
    return numberOfNewConnections;
  }

  static class AirportNode {
    String airport;
    List<String> connections;
    boolean isReachable;
    List<String> unreachableConnections;

    public AirportNode(String airport) {
      this.airport = airport;
      connections = new ArrayList<String>();
      isReachable = true;
      unreachableConnections = new ArrayList<String>();
    }
    }
}

```