Solution 1

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

// 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
function airportConnections(airports, routes, startingAirport) {
  const airportGraph = createAirportGraph(airports, routes);
  const unreachableAirportNodes = getUnreachableAirportNodes(airportGraph, airports, startingAirport);
  markUnreachableConnections(airportGraph, unreachableAirportNodes);
  return getMinNumberOfNewConnections(airportGraph, unreachableAirportNodes);
}

// O(a + r) time | O(a + r) space
function createAirportGraph(airports, routes) {
  const airportGraph = {};
  for (const airport of airports) {
    airportGraph[airport] = new AirportNode(airport);
  }
  for (const route of routes) {
    const [airport, connection] = route;
    airportGraph[airport].connections.push(connection);
  }
  return airportGraph;
}

// O(a + r) time | O(a) space
function getUnreachableAirportNodes(airportGraph, airports, startingAirport) {
  const visitedAirports = {};
  depthFirstTraverseAirports(airportGraph, startingAirport, visitedAirports);

  const unreachableAirportNodes = [];
  for (const airport of airports) {
    if (airport in visitedAirports) continue;
    const airportNode = airportGraph[airport];
    airportNode.isReachable = false;
    unreachableAirportNodes.push(airportNode);
  }
  return unreachableAirportNodes;
}

function depthFirstTraverseAirports(airportGraph, airport, visitedAirports) {
  if (airport in visitedAirports) return;
  visitedAirports[airport] = true;
  const {connections} = airportGraph[airport];
  for (const connection of connections) {
    depthFirstTraverseAirports(airportGraph, connection, visitedAirports);
  }
}

// O(a * (a + r)) time | O(a) space
function markUnreachableConnections(airportGraph, unreachableAirportNodes) {
  for (const airportNode of unreachableAirportNodes) {
    const {airport} = airportNode;
    const unreachableConnections = [];
    depthFirstAddUnreachableConnections(airportGraph, airport, unreachableConnections, {});
    airportNode.unreachableConnections = unreachableConnections;
  }
}

function depthFirstAddUnreachableConnections(airportGraph, airport, unreachableConnections, visitedAirports) {
  if (airportGraph[airport].isReachable) return;
  if (airport in visitedAirports) return;
  visitedAirports[airport] = true;
  unreachableConnections.push(airport);
  const {connections} = airportGraph[airport];
  for (const connection of connections) {
    depthFirstAddUnreachableConnections(airportGraph, connection, unreachableConnections, visitedAirports);
  }
}

// O(alog(a) + a + r) time | O(1) space
function getMinNumberOfNewConnections(airportGraph, unreachableAirportNodes) {
  unreachableAirportNodes.sort((a1, a2) => a2.unreachableConnections.length - a1.unreachableConnections.length);

  let numberOfNewConnections = 0;
  for (const airportNode of unreachableAirportNodes) {
    if (airportNode.isReachable) continue;
    numberOfNewConnections++;
    for (const connection of airportNode.unreachableConnections) {
      airportGraph[connection].isReachable = true;
    }
  }
  return numberOfNewConnections;
}

class AirportNode {
  constructor(airport) {
    this.airport = airport;
    this.connections = [];
    this.isReachable = true;
    this.unreachableConnections = [];
  }
}

exports.airportConnections = airportConnections;

```