Solution 1

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

#include <vector>
#include <unordered_map>
#include <algorithm>
using namespace std;

class AirportNode {
public:
  string airport;
  vector<string> connections;
  bool isReachable;
  vector<string> unreachableConnections;

  AirportNode(string airport) {
    this->airport = airport;
    connections = {};
    isReachable = true;
    unreachableConnections = {};
  }
};

unordered_map<string, AirportNode *>
createAirportGraph(vector<string> airports, vector<vector<string>> routes);
vector<AirportNode *>
getUnreachableAirportNodes(unordered_map<string, AirportNode *> airportGraph,
                           vector<string> airports, string startingAirport);
void depthFirstTraverseAirports(
    unordered_map<string, AirportNode *> airportGraph, string airport,
    unordered_map<string, bool> *visitedAirports);
void markUnreachableConnections(
    unordered_map<string, AirportNode *> airportGraph,
    vector<AirportNode *> unreachableAirportNodes);
void depthFirstAddUnreachableConnections(
    unordered_map<string, AirportNode *> airportGraph, string airport,
    vector<string> *unreachableConnections,
    unordered_map<string, bool> *visitedAirports);
int getMinNumberOfNewConnections(
    unordered_map<string, AirportNode *> airportGraph,
    vector<AirportNode *> unreachableAirportNodes);

// 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
int airportConnections(vector<string> airports, vector<vector<string>> routes,
                       string startingAirport) {
  unordered_map<string, AirportNode *> airportGraph =
      createAirportGraph(airports, routes);
  vector<AirportNode *> unreachableAirportNodes =
      getUnreachableAirportNodes(airportGraph, airports, startingAirport);
  markUnreachableConnections(airportGraph, unreachableAirportNodes);
  return getMinNumberOfNewConnections(airportGraph, unreachableAirportNodes);
}

// O(a + r) time | O(a + r) space
unordered_map<string, AirportNode *>
createAirportGraph(vector<string> airports, vector<vector<string>> routes) {
  unordered_map<string, AirportNode *> airportGraph = {};
  for (string airport : airports) {
    airportGraph[airport] = new AirportNode(airport);
  }
  for (vector<string> route : routes) {
    string airport = route[0];
    string connection = route[1];
    airportGraph[airport]->connections.push_back(connection);
  }
  return airportGraph;
}

// O(a + r) time | O(a) space
vector<AirportNode *>
getUnreachableAirportNodes(unordered_map<string, AirportNode *> airportGraph,
                           vector<string> airports, string startingAirport) {
  unordered_map<string, bool> visitedAirports = {};
  depthFirstTraverseAirports(airportGraph, startingAirport, &visitedAirports);

  vector<AirportNode *> unreachableAirportNodes = {};
  for (string airport : airports) {
    if (visitedAirports.find(airport) != visitedAirports.end())
      continue;
    AirportNode *airportNode = airportGraph[airport];
    airportNode->isReachable = false;
    unreachableAirportNodes.push_back(airportNode);
  }
  return unreachableAirportNodes;
}

void depthFirstTraverseAirports(
    unordered_map<string, AirportNode *> airportGraph, string airport,
    unordered_map<string, bool> *visitedAirports) {
  if (visitedAirports->find(airport) != visitedAirports->end())
    return;
  visitedAirports->insert({airport, true});
  vector<string> connections = airportGraph[airport]->connections;
  for (string connection : connections) {
    depthFirstTraverseAirports(airportGraph, connection, visitedAirports);
  }
}

// O(a * (a + r)) time | O(a) space
void markUnreachableConnections(
    unordered_map<string, AirportNode *> airportGraph,
    vector<AirportNode *> unreachableAirportNodes) {
  for (AirportNode *airportNode : unreachableAirportNodes) {
    string airport = airportNode->airport;
    vector<string> unreachableConnections = {};
    unordered_map<string, bool> visitedAirports = {};
    depthFirstAddUnreachableConnections(
        airportGraph, airport, &unreachableConnections, &visitedAirports);
    airportNode->unreachableConnections = unreachableConnections;
  }
}

void depthFirstAddUnreachableConnections(
    unordered_map<string, AirportNode *> airportGraph, string airport,
    vector<string> *unreachableConnections,
```

```
    unordered_map<string, bool> *visitedAirports) {
  if (airportGraph[airport]->isReachable)
    return;
  if (visitedAirports->find(airport) != visitedAirports->end())
    return;
  visitedAirports->insert({airport, true});
  unreachableConnections->push_back(airport);
  vector<string> connections = airportGraph[airport]->connections;
  for (string connection : connections) {
    depthFirstAddUnreachableConnections(
        airportGraph, connection, unreachableConnections, visitedAirports);
  }
}

// O(alog(a) + a + r) time | O(1) space
int getMinNumberOfNewConnections(
    unordered_map<string, AirportNode *> airportGraph,
    vector<AirportNode *> unreachableAirportNodes) {
  sort(unreachableAirportNodes.begin(), unreachableAirportNodes.end(),
       [](AirportNode *a1, AirportNode *a2) -> bool {
         return a2->unreachableConnections.size() <
                a1->unreachableConnections.size();
       });

  int numberOfNewConnections = 0;
  for (AirportNode *airportNode : unreachableAirportNodes) {
    if (airportNode->isReachable)
      continue;
    numberOfNewConnections++;
    for (string connection : airportNode->unreachableConnections) {
      airportGraph[connection]->isReachable = true;
    }
  }
  return numberOfNewConnections;
}
```