

Solution 1

```
1 // Copyright © 2020 AlgoExpert, LLC. All rights reserved.
2
3 #include <vector>
4 #include <unordered_map>
5 #include <algorithm>
6 using namespace std;
7
8 class AirportNode {
9 public:
10     string airport;
11     vector<string> connections;
12     bool isReachable;
13     vector<string> unreachableConnections;
14
15     AirportNode(string airport) {
16         this->airport = airport;
17         connections = {};
18         isReachable = true;
19         unreachableConnections = {};
20     }
21 };
22
23 unordered_map<string, AirportNode *>
24 createAirportGraph(vector<string> airports, vector<vector<string>> routes);
25 vector<AirportNode *>
26 getUnreachableAirportNodes(unordered_map<string, AirportNode *> airportGraph,
27                             vector<string> airports, string startingAirport);
28 void depthFirstTraverseAirports(
29     unordered_map<string, AirportNode *> airportGraph, string airport,
30     unordered_map<string, bool> *visitedAirports);
31 void markUnreachableConnections(
32     unordered_map<string, AirportNode *> airportGraph,
33     vector<AirportNode *> unreachableAirportNodes);
34 void depthFirstAddUnreachableConnections(
35     unordered_map<string, AirportNode *> airportGraph, string airport,
36     vector<string> *unreachableConnections,
37     unordered_map<string, bool> *visitedAirports);
38 int getMinNumberOfNewConnections(
39     unordered_map<string, AirportNode *> airportGraph,
40     vector<AirportNode *> unreachableAirportNodes);
41
42 // O(a * (a + r) + a + r + alog(a)) time | O(a + r) space - where a is the
43 // number of airports and r is the number of routes
44 int airportConnections(vector<string> airports, vector<vector<string>> routes,
45                         string startingAirport) {
46     unordered_map<string, AirportNode *> airportGraph =
47         createAirportGraph(airports, routes);
48     vector<AirportNode *> unreachableAirportNodes =
49         getUnreachableAirportNodes(airportGraph, airports, startingAirport);
50     markUnreachableConnections(airportGraph, unreachableAirportNodes);
51     return getMinNumberOfNewConnections(airportGraph, unreachableAirportNodes);
52 }
53
54 // O(a + r) time | O(a + r) space
55 unordered_map<string, AirportNode *>
56 createAirportGraph(vector<string> airports, vector<vector<string>> routes) {
57     unordered_map<string, AirportNode *> airportGraph = {};
58     for (string airport : airports) {
59         airportGraph[airport] = new AirportNode(airport);
60     }
61     for (vector<string> route : routes) {
62         string airport = route[0];
63         string connection = route[1];
64         airportGraph[airport]->connections.push_back(connection);
65     }
66     return airportGraph;
67 }
68
69 // O(a + r) time | O(a) space
70 vector<AirportNode *>
71 getUnreachableAirportNodes(unordered_map<string, AirportNode *> airportGraph,
72                             vector<string> airports, string startingAirport) {
73     unordered_map<string, bool> visitedAirports = {};
74     depthFirstTraverseAirports(airportGraph, startingAirport, &visitedAirports);
75
76     vector<AirportNode *> unreachableAirportNodes = {};
77     for (string airport : airports) {
78         if (visitedAirports.find(airport) != visitedAirports.end())
79             continue;
80         AirportNode *airportNode = airportGraph[airport];
81         airportNode->isReachable = false;
82         unreachableAirportNodes.push_back(airportNode);
83     }
84     return unreachableAirportNodes;
85 }
86
87 void depthFirstTraverseAirports(
88     unordered_map<string, AirportNode *> airportGraph, string airport,
89     unordered_map<string, bool> *visitedAirports) {
90     if (visitedAirports->find(airport) != visitedAirports->end())
91         return;
92     visitedAirports->insert({airport, true});
93     vector<string> connections = airportGraph[airport]->connections;
94     for (string connection : connections) {
95         depthFirstTraverseAirports(airportGraph, connection, visitedAirports);
96     }
97 }
98
99 // O(a * (a + r)) time | O(a) space
100 void markUnreachableConnections(
101     unordered_map<string, AirportNode *> airportGraph,
102     vector<AirportNode *> unreachableAirportNodes) {
103     for (AirportNode *airportNode : unreachableAirportNodes) {
104         string airport = airportNode->airport;
105         vector<string> unreachableConnections = {};
106         unordered_map<string, bool> visitedAirports = {};
107         depthFirstAddUnreachableConnections(
108             airportGraph, airport, &unreachableConnections, &visitedAirports);
109         airportNode->unreachableConnections = unreachableConnections;
110     }
111 }
112
113 void depthFirstAddUnreachableConnections(
114     unordered_map<string, AirportNode *> airportGraph, string airport,
115     vector<string> *unreachableConnections,
```

```
116     unordered_map<string, bool> *visitedAirports) {
117     if (airportGraph[airport]->isReachable)
118         return;
119     if (visitedAirports->find(airport) != visitedAirports->end())
120         return;
121     visitedAirports->insert({airport, true});
122     unreachableConnections->push_back(airport);
123     vector<string> connections = airportGraph[airport]->connections;
124     for (string connection : connections) {
125         depthFirstAddUnreachableConnections(
126             airportGraph, connection, unreachableConnections, visitedAirports);
127     }
128 }
129
130 // O(a log(a) + a + r) time | O(1) space
131 int getMinNumberOfNewConnections(
132     unordered_map<string, AirportNode *> airportGraph,
133     vector<AirportNode *> unreachableAirportNodes) {
134     sort(unreachableAirportNodes.begin(), unreachableAirportNodes.end(),
135         [](AirportNode *a1, AirportNode *a2) -> bool {
136             return a2->unreachableConnections.size() <
137                 a1->unreachableConnections.size();
138         });
139
140     int numberOfNewConnections = 0;
141     for (AirportNode *airportNode : unreachableAirportNodes) {
142         if (airportNode->isReachable)
143             continue;
144         numberOfNewConnections++;
145         for (string connection : airportNode->unreachableConnections) {
146             airportGraph[connection]->isReachable = true;
147         }
148     }
149     return numberOfNewConnections;
150 }
```