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

Prompt　Scratchpad　Our Solution(s)　Video Explanation　Run Code

Solution 1

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

package main

import "sort"

type AirportNode struct {
  Airport                 string
  Connections             []string
  IsReachable             bool
  UnreachableConnections []string
}

func NewAirportNode(airport string) *AirportNode {
  return &AirportNode{
    Airport:                airport,
    Connections:            []string{},
    IsReachable:            true,
    UnreachableConnections: []string{},
  }
}

// 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
func AirportConnections(airports []string, routes [][]string, startingAirport string) int {
  airportGraph := createAirportGraph(airports, routes)
  unreachableAirportNodes := getUnreachableAirportNodes(airportGraph, airports, startingAirport)
  markUnreachableConnections(airportGraph, unreachableAirportNodes)
  return getMinNumberOfNewConnections(airportGraph, unreachableAirportNodes)
}

// O(a + r) time | O(a + r) space
func createAirportGraph(airports []string, routes [][]string) map[string]*AirportNode {
  airportGraph := map[string]*AirportNode{}
  for _, airport := range airports {
    airportGraph[airport] = NewAirportNode(airport)
  }
  for _, route := range routes {
    airport, connection := route[0], route[1]
    airportGraph[airport].Connections = append(airportGraph[airport].Connections, connection)
  }
  return airportGraph
}

// O(a + r) time | O(a) space
func getUnreachableAirportNodes(
  airportGraph map[string]*AirportNode, airports []string, startingAirport string,
) []*AirportNode {
  visitedAirports := map[string]bool{}
  depthFirstTraverseAirports(airportGraph, startingAirport, visitedAirports)

  unreachableAirportNodes := []*AirportNode{}
  for _, airport := range airports {
    if _, found := visitedAirports[airport]; found {
      continue
    }
    airportNode := airportGraph[airport]
    airportNode.IsReachable = false
    unreachableAirportNodes = append(unreachableAirportNodes, airportNode)
  }
  return unreachableAirportNodes
}

func depthFirstTraverseAirports(
  airportGraph map[string]*AirportNode, airport string, visitedAirports map[string]bool,
) {
  if _, found := visitedAirports[airport]; found {
    return
  }
  visitedAirports[airport] = true
  connections := airportGraph[airport].Connections
  for _, connection := range connections {
    depthFirstTraverseAirports(airportGraph, connection, visitedAirports)
  }
}

// O(a * (a + r)) time | O(a) space
func markUnreachableConnections(
  airportGraph map[string]*AirportNode, unreachableAirportNodes []*AirportNode,
) {
  for _, airportNode := range unreachableAirportNodes {
    airport := airportNode.Airport
    unreachableConnections := []string{}
    visitedAirports := map[string]bool{}
    depthFirstAddUnreachableConnections(airportGraph, airport, &unreachableConnections, visitedAirports)
    airportNode.UnreachableConnections = unreachableConnections
  }
  return
}

func depthFirstAddUnreachableConnections(
  airportGraph map[string]*AirportNode, airport string,
  unreachableConnections *[]string, visitedAirports map[string]bool,
) {
  if airportGraph[airport].IsReachable {
    return
  } else if _, found := visitedAirports[airport]; found {
    return
  }
  visitedAirports[airport] = true
  *unreachableConnections = append(*unreachableConnections, airport)
  connections := airportGraph[airport].Connections
  for _, connection := range connections {
    depthFirstAddUnreachableConnections(airportGraph, connection, unreachableConnections, visitedAirports)
  }
}

// O(alog(a) + a + r) time | O(1) space
func getMinNumberOfNewConnections(
  airportGraph map[string]*AirportNode, unreachableAirportNodes []*AirportNode,
) int {
  sort.SliceStable(unreachableAirportNodes, func(i, j int) bool {
    a1, a2 := unreachableAirportNodes[i], unreachableAirportNodes[j]
    return len(a1.UnreachableConnections) > len(a2.UnreachableConnections)
  })
  numberOfNewConnections := 0
  for _, node := range unreachableAirportNodes {
    if node.IsReachable {
      continue
    }
    numberOfNewConnections++
    for _, connection := range node.UnreachableConnections {
      airportGraph[connection].IsReachable = true
    }
  }
  return numberOfNewConnections
}
```