Algorithm Analysis
CSI 5301

Assignment 5.1: The UN Problem
Due: Mon June 13, 2022 11:59pm
Submitted by: Pawan Bhandari

Description:

Read from a input file that contains the following information:

Input starts with a positive integer, $1 \leq N \leq 100$, the number of members in the meeting. This is followed by N lines, each line describing a member. Each of these N lines starts with the member's name (which is distinct), then the language(s) that the member speaks (this will only be a single language), then a list of 0 to 20 additional languages the member understands but doesn't speak. All members understand the language they speak. All members and language names are sequences of 1 to 15 letters (a-z and A-Z), numbers, and hyphens. Member names and languages are separated by single spaces.

For example:
Ian Italian French Spanish
In this case, the name would be "Ian," who speaks "Italian" and understands "French" and "Spanish." As mentioned previously, there will always be a single language that is spoken (the one right after the name). Any additional languages they understand will be after that first spoken language, although they might not understand any other language.

After processing the input file, print the minimum number of members that will be required to leave so that all remaining members can converse with each other in a meeting.

Data Structure:

Graph
vector<string>, vector<Member>, set<string>, stack<Member>, bool[]

Algorithm:

1) Read input.txt to populate memberVector and find number of members

```
readInputFile (vector<Member> *memberVector, string inputFileName)
```

2) Create a Adjacency list representation Graph with members as vertices and add a directional edge from member1 to member2 if member2 can understand a language that member1 speaks

```
createMemberGraph (vector<Member> *memberVector, MemberGraph
                   *UNMemberGraph, ofstream &outputData)

start

end
```

3) Apply Kosaraju's algorithm on the MemberGraph to find a list of strongly connected members (i.e. group of members who can converse with each other)

MemberGraph::findSCC()

start

end

4) Calculate minimum number of members required to leave

start

maxSccMembersSize = the size of largest SCC of members from step 3.
numberOfMembers = total number of members (calculated in step 1).
membersRequiredToLeave = minimum number of members required to leave.

membersRequiredToLeave = numberOfMembers – maxSccMembersSize

print membersRequiredToLeave as output to the terminal.

end

Analysis:

Input N	int numberOfMembers Vector<Members> memberVector (will be implicitly derived from input.txt)
Basic Operation	Depth-First-Search (DFS) on a Graph Kosaraju's Algorithm Implementation Vector operations Recursion
Recurrence relation	Best Case: $T(n)=TODO$ Worst Case: $T(n)=TODO$

Worst Case Analysis:

$O(V^2)$for populating the MemberGraph with correct edges from the input file.

$O(V+E)$To determine the output by using Kosaraju's Algorithm for finding SCCs.

Best Case Analysis: